Über die Stabilität von parametrisierten algebraischen Spezifikationen

Diplomarbeit
im Fach Informatik

an der
Universität Dortmund

vorgelegt von
Hubert Baumeister[1]
Dortmund, Juli 1990

[1]Zur Zeit am Max-Planck Institut für Informatik, Im Stadtwald, D-6600 Saarbrücken,
E-mail: Hubert.Baumeister@cs.uni-sb.de

Inhaltsverzeichnis

Einleitung

Bei der Spezifikation von Software ergibt sich öfters das Problem, daß man Implementierungen von Spezifikationen zulassen möchte, obwohl sie nicht die Axiome der Spezifikation exakt erfüllen. Diese Implementierungen lassen sich aber für den Benutzer nicht von einer, die die Spezifikation exakt erfüllt, unterscheiden.

Ein Beispiel dafür ist die Implementierung einer Spezifikation eines Stacks über beliebige Elemente durch ein Feld–Zeiger Paar [Sch86]. Die Spezifikation Stack enhält die üblichen Operationen auf Stacks

- $empty$: $stack$
- $push$: $stack \times elem \rightarrow stack$
- pop: $stack \rightarrow stack$
- top: $stack \rightarrow elem$

und die üblichen Axiome

$$\begin{aligned} pop(push(s,e)) &= s \\ top(push(s,e)) &= e. \end{aligned}$$

Bei der Realisierung durch ein Feld–Zeiger Paar enthält das Feld die Elemente des Stacks und der Zeiger zeigt auf das oberste Element des Stacks. Die $push$–Operation erhöht den Zeiger um eins und legt das Element in dem Feld ab, auf das der Zeiger zeigt. Bei der pop–Operation wird der Zeiger einfach um eins erniedrigt. Für den Benutzer des Stacks zeigt die Realisierung durch ein Feld–Zeiger Paar das gewünschte Verhalten eines Stacks, nämlich die Elemente in der umgekehrter Reihenfolge vom Stack holen zu können, in der sie dort abgelegt wurden. Aber diese Realsierung eines Stacks erfüllt nicht das Axiom $pop(push(s,e)) = s$, da bei der $push$–Operation ein Element in einem Feld überschrieben wird, was bei der pop–Operation nicht restauriert werden kann.

Das Problem entsteht durch die Gleichheitsoperation auf Stacks, da sie die Stacks bzgl. ihrer internen Darstellung vergleicht. Für den Benutzer eines Stacks dagegen ist es egal wie der Stack tatsächlich aussieht, wenn er nur das gewünschte Verhalten zeigt. Für ihn sind die Elemente des Stacks nicht beobachtbar. Diese Abstraktion von der internen Darstellung ist der Kern der Datenabstraktion.

Die Gleichheit bei der Spezifikation zu verbieten, ist keine Lösung des Problems, da sich das gewünschte Verhalten von Stacks selbst mit Formeln der Prädikatenlogik 1. Stufe nicht endlich spezifizieren läßt [Sch86].

Um dennoch Implementierungen zuzulassen, die eine Spezifikation zwar nicht exakt erfüllen (z.B. optimierende Implementierungen [Gan83]), die aber für den Benutzer das gleiche Verhalten haben wie Implementierungen, die die Spezifikation exakt erfüllen, wird der Begriff der

Abstraktorimplementierung [ST86] eingeführt. Eine Abstraktorimplementierung ist eine Implementierung nicht der Spezifikation selbst, sondern des Verhaltens einer Spezifikation.

Bei parametrisierte Spezifikationen tritt das Problem auf, daß sie nur Eigenschaften aus dem Verhalten eines aktuellen Parameters verwenden dürfen. Tun sie das nicht, wird das Ergebnis der Anwendung abhängig von Eigenschaften, die nicht jeder aktuelle Parameter erfüllt. Verwendet eine parametrisierte Spezifikation nur die Eigenschaften aus dem Verhalten eines aktuellen Parameters, so wird sie stabil genannt.

Im ersten Teil der Arbeit werden die Grundlagen, die in dieser Arbeit benötigt werden, vorgestellt. Das erste Kapitel ist eine kurze Zusammenfassung von Definitionen aus der Kategorientheorie. Der Leser, der z.B. mit dem Begriff Adjunktion vertraut ist, kann das Kapitel ruhig überspringen. Im zweiten Kapitel wird der Begriff einer Institution [GB90] vorgestellt, der den geeigneten Rahmen für eine von der konkreten Logik unabhängigen Darstellung liefert. Für den Abschnitt über die Verhaltensäquivalenz wird der Begriff von freien Variablen in Institutionen benötigt.

Der zweite Teil der Arbeit befaßt sich mit parametrisierten abstrakten Datentypen und Spezifikationen. Es wird versucht, unabhängig von einer konkreten Logik, die lose Semantik von Spezifikationen [ST85b, Wir89] mit der initialen Semantik zu verbinden. Speziell wird versucht die freie Funktorsemantik von parametrisierten Spezifikationen [EM85] als Spezialfall der losen Semantik darzustellen.

Der dritte Teil befaßt sich mit der Abstraktorimplementierung und deren Kompositionseigenschaften. Besonderer Wert wird auf die Zulässigkeit von aktuellen Parameter bzgl. eines Abstraktors und auf die Stabilität von parametrisierten Spezifikationen gelegt.

Bedanken möchte ich mich bei Prof. Dr. Harald Ganzinger für die Betreuung der Diplomarbeit und bei Dr. Hagen Huwig für anregende Diskussionen.

Teil I

Grundlagen

Kapitel 1

Kategorientheorie

In diesem Kapitel werden einige Grundbegriffe der Kategorientheorie aus [Mac88] vorgestellt, die in dieser Arbeit verwendet werden.

Definition 1.1 (Kategorie) *Eine* Katagorie $\mathbf{C}$ *besteht aus einer Klasse* $|\mathbf{C}|$ *von* Objekten, *einer Klasse* $arr(\mathbf{C})$ *von* Morphismen *oder Pfeilen (arrows), zwei Operationen* quelle *und* ziel, *die Morphismen auf Objekte abbilden und einer Operation* $\circ$, *die zu zwei Morphismen die Komposition der beiden Morphismen liefert. Außerdem gelten folgende Gesetze:*

1. *(Komposition) Für Morphismen* f *und* g *aus* $arr(\mathbf{C})$ *mit* $quelle(f) = ziel(g)$ *muß* $f \circ g$ *definiert sein und einen Morphismus aus* $arr(\mathbf{C})$ *liefern.*

2. *(Assoziativität) Ist die Komposition von* $f \circ g$ *und von* $g \circ h$ *für Morphismen* f, g *und* h *aus* $arr(\mathbf{C})$ *definiert, dann ist*

$$f \circ (g \circ h) = (f \circ g) \circ h.$$

3. *(Identität) Für jedes Objekt* c *aus* $|\mathbf{C}|$ *gibt es einen eindeutigen Morphismus* id_c *aus* $arr(\mathbf{C})$, *für den gilt*

$$id_c \circ f = f$$

und

$$g \circ id_c = g.$$

Für einen Morphismus f aus $arr(\mathbf{C})$ mit $quelle(f) = c$ und $ziel(f) = d$, wird auch $f\colon c \to d$ geschrieben. Für die Komposition $f \circ g$ von zwei Morphismen wird häufig nur fg geschrieben. Kategorien werden in Großbuchstaben und fett geschrieben. Ist aus dem Zusammenhang klar ob es sich um Morphismen f oder Objekte c handelt, dann wird statt $f \in arr(\mathbf{C})$ auch $f \in \mathbf{C}$ und statt $c \in |\mathbf{C}|$ auch $c \in \mathbf{C}$ geschrieben.

Beispiele für Kategorie sind die leere Kategorie $\mathbf{0}$, die keine Objekte und keine Morphismen enthält, die Kategorie $\mathbf{1}$ mit genau einem Objekt und einem Morphismus und die Kategorie $\mathbf{2}$, die zwei Objekte enthält und nur einen Morphismus, der nicht die Identität ist.

Definition 1.2 (Unterkategorie) *Eine* Unterkategorie $\mathbf{D}$ *einer Kategorie* $\mathbf{C}$

$$\mathbf{D} \subseteq \mathbf{C}$$

ist eine Kategorie, bei der die Morphismen und Objekte Teilklassen der Morphismen und Objekte von $\mathbf{C}$ *sind.*

Eine Unterkategorie $\mathbf{D}$ *heißt* voll, *wenn zu zwei beliebigen Objekten aus* $\mathbf{D}$ *alle Morphismen zwischen diesen Objekten aus* $\mathbf{C}$ *in* $\mathbf{D}$ *enthalten sind.*

Eine volle Unterkategorie $\mathbf{D}$ einer Kategorie $\mathbf{C}$ ist damit eindeutig durch die Angabe ihrer Objekte $|\mathbf{D}|$ bestimmt. Tritt im folgenden eine Teilmenge M von Objekten einer Kategorie $\mathbf{C}$ an einer Stelle auf, an der eine Kategorie stehen müsste, so ist immer die volle, durch die Objekte in M bestimmte, Unterkategorie von $\mathbf{C}$ gemeint.

Definition 1.3 (Initiale Objekte) *Ein Objekt c einer Kategorie $\mathbf{C}$ heißt initial, gdw. es zu jedem Objekt d in $\mathbf{C}$ genau einen Morphismus $f: c \to d$ in $arr(\mathbf{C})$ gibt.*

Die Existenz eines initialen Objekts einer Kategorie ist nicht immer gegeben. Existiert aber ein initiales Objekt, so ist es nur bis auf Isomorphie bestimmt.

Definition 1.4 (Durchschnitt von Kategorien) *Der Durchschnitt von zwei Kategorien $\mathbf{D}$ und $\mathbf{C}$*

$$\mathbf{D} \cap \mathbf{C}.$$

ist der Durchschnitt der Objekte und Morphismen von $\mathbf{D}$ und $\mathbf{C}$

$$|\mathbf{D} \cap \mathbf{C}| = |\mathbf{C}| \cap |\mathbf{D}|$$

$$arr(\mathbf{D} \cap \mathbf{C}) = \mathbf{arr}(\mathbf{C}) \cap \mathbf{arr}(\mathbf{D}).$$

Es ist leicht zu zeigen, daß $\mathbf{D} \cap \mathbf{C}$ eine Kategorie ist. Sind $\mathbf{C}$ und $\mathbf{D}$ volle Unterkategorien einer Kategorie $\mathbf{E}$, dann ist $\mathbf{C} \cap \mathbf{D}$ ebenfalls eine volle Unterkategorie von $\mathbf{E}$.

Definition 1.5 (Funktor) *Zu zwei Kategorien $\mathbf{C}$ und $\mathbf{D}$ besteht ein* Funktor $F: \mathbf{C} \to \mathbf{D}$ *aus einer Funktion F, die Objekte aus $\mathbf{C}$ auf Objekte aus $\mathbf{D}$ abbildet und einer Funktion (ebenfalls F genannt), die Morphismen aus $\mathbf{C}$ auf Morphismen aus $\mathbf{D}$ abbildet, wobei F die Identitäten erhält und mit der Komposition von Morphismen verträglich ist. Das heißt, ist id_c die Identität von c in $\mathbf{C}$, dann ist $F(id_c)$ die Identiät von $F(c)$ in $\mathbf{D}$*

$$F(id_c) = id_{F(c)}.$$

Ist gh in $\mathbf{C}$ definiert, dann ist $F(g)G(h)$ in $\mathbf{D}$ definiert und es gilt

$$F(gh) = F(g)F(h).$$

Ein Funktor $V: \mathbf{C} \to \mathbf{D}$, der Strukturen der Kategorie $\mathbf{C}$ vergißt, wird auch Vergißfunktor genannt. Ein Beispiel eines Vergißfunktors ist der Reduktfunktor $Alg(\sigma)$ zu dem Signaturmorphismus $\sigma: \Sigma \to \Sigma'$ in der Gleichungslogik (s.a. Definition 2.6). Zu einer Algebra A aus $Alg(\Sigma')$ vergißt $Alg(\sigma)$ alle Operationen, die in Σ', aber nicht in Σ vorkommen und alle Trägermengen zu Sorten, die in Σ', aber nicht in Σ vorkommen.

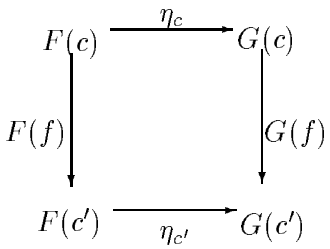
Der Funktor $Id_C: \mathbf{C} \to \mathbf{C}$, der Objekte und Morphismen aus $\mathbf{C}$ auf sich selbst abbildet ist der Identitätsfunktor auf Kategorien. Die Klasse aller kleinen Kategorien zusammen mit der Klasse aller Funktoren zwischen Kategorien bilden wieder eine Kategorie, die Kategorie der kleinen Kategorien **Cat** (s.a. [Mac88] für eine Diskussion der mathematischen Grundlagen für diese Kategorie).

Zu einer Kategorie $\mathbf{C}$ ist die Kategorie $\mathbf{C^{op}}$ definiert, die alle Objekte von $\mathbf{C}$ und genau die Morphismen $f: d \to c$ enthält, für die ein Morphismus $f: c \to d$ in $\mathbf{C}$ existiert. Das heißt in der Kategorie $\mathbf{C^{op}}$ sind alle Pfeile der Kategorie $\mathbf{C}$ umgedreht.

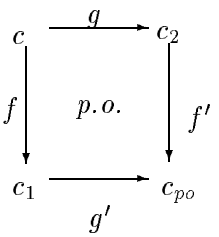
Definition 1.6 (Natürliche Transformation) *Sind zwei Funktoren F und G von* **C** *nach* **D** *gegeben, dann ist eine natürliche Transformation η von F nach G*

$$\eta: F \xrightarrow{\bullet} G$$

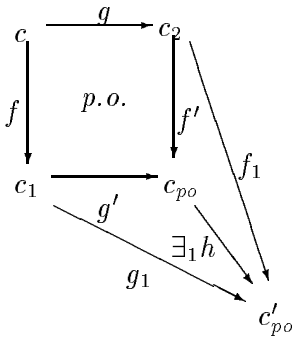
eine Funktion, die zu jedem c aus **C** *einen Morphismus $\eta_c: F(c) \to G(c)$ liefert, so daß für jeden Morphismus $f: c \to c'$ aus* **C** *das folgende Diagramm kommutiert.*



Definition 1.7 (Pushout) *Ein Pushout zu einem Diagramm $D = c_1 \xleftarrow{f} c \xrightarrow{g} c_2$ in einer Kategorie* **C** *ist eine universelle Erweiterung $E = c_1 \xrightarrow{g'} c_{po} \xleftarrow{f'} c_2$ des Diagramms zu einem kommutierenden Rechteck.*



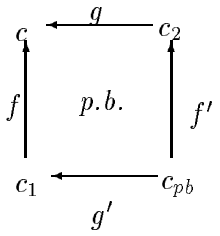
Das heißt, gibt es eine weitere Erweiterung $E' = c_1 \xrightarrow{g_1} c'_{po} \xleftarrow{f_1} c_2$ des Diagramms D zu einem kommutierenden Rechteck, dann existiert genau ein Morphismus $h: c_{po} \to c'_{po}$ mit hg' gleich g_1 und hf' gleich f_1.



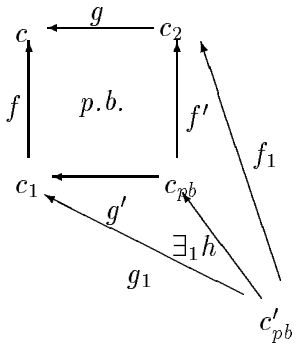
Falls ein Pushout zu einem Diagramm in **C** existiert, ist er bis auf Isomorphie eindeutig bestimmt.

Der Begriff Pullback ist dual zu dem Begriff des Pushouts. Dreht man die Richtung der Morphismen in der Definition des Pushouts um, so erhält man die Definition eines Pullbacks.

Definition 1.8 (Pullback) *Ein Pullback zu einem Diagramm* $D = c_1 \xrightarrow{f} c \xleftarrow{g} c_2$ *in einer Kategorie* **C** *ist eine universelle Erweiterung* $E = c_1 \xleftarrow{g'} c_{pb} \xrightarrow{f'} c_2$ *des Diagramms zu einem kommutierenden Rechteck*

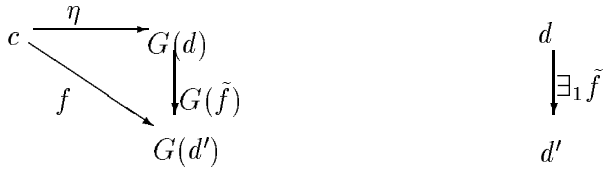


Das heißt, gibt es eine weitere Erweiterung $E' = c_1 \xleftarrow{g_1} c'_{pb} \xrightarrow{f_1} c_2$ *des Diagramms* D *zu einem kommutierenden Rechteck, dann existiert genau ein Morphismus* $h: c'_{pb} \to c_{pb}$ *mit* $g'h$ *gleich* g_1 *und* $f'h$ *gleich* f_1.



Falls ein Pullback zu einem Diagramm in **C** existiert, ist er bis auf Isomorphie eindeutig bestimmt.

Definition 1.9 (Universeller Morphismus) *Ist* $G: \mathbf{D} \to \mathbf{C}$ *ein Funktor, dann besteht ein universeller Morphismus* $\langle c, \eta: c \to G(d)\rangle$ *aus einem Objekt* c*, einem Morphismus* η *aus* **C** *und einem Objekt* d *aus* **D** *und es gilt für alle Objekte* d' *aus* **D***, für die es einen Morphismus* $f: c \to G(d')$ *gibt, es existiert genau ein Morphismus* $\tilde{f}: d \to d'$ *mit* $G(\tilde{f})\eta = f$.

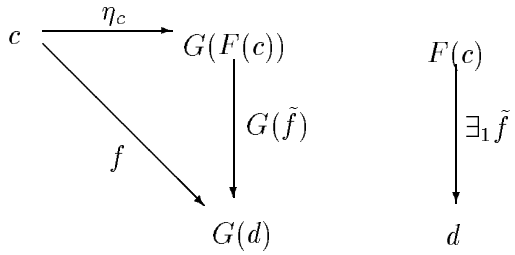


Die Eigenschaften eines universellen Morphismus $\langle c, \eta: c \to G(d)\rangle$ spielen eine zentrale Rolle bei der Semantik parametrisierter Spezifikationen in [EM85], weil aus ihnen folgt, daß wenn c ein initiales Objekt in der Kategorie **C** ist, d ein initiales Objekt in **D** ist.

Satz 1.10 *Ist* c *ein initiales Objekt in* **C**, G *ein Funktor von* **D** *nach* **C** *und* $\langle c, \eta: c \to G(d)\rangle$ *ein universeller Morphismus, dann ist* d *ein initiales Objekt in* **D**.

Beweis Für ein Objekt $d' \in \mathbf{C}$ ist zu zeigen, daß es einen eindeutigen Morphismus von d nach d' gibt. Da c initiales Objekt in $\mathbf{C}$ ist, existiert ein Morphismus f von c nach $G(d')$ und damit gibt es einen eindeutigen Morphismus $\tilde{f}$ von d nach d'. Der Morphismus $\tilde{f}$ ist auch einziger Morphismus von d nach d', da wegen der Initialität von c der Morphismus $f: c \to G(d')$ eindeutig ist. ■

Definition 1.11 (Adjunktion) *Eine Adjunktion $\langle F, G, \eta \rangle: C \to D$, besteht aus zwei Funktoren $F: C \to D$ und $G: D \to C$ und einer natürlichen Transformation $\eta: Id_C \xrightarrow{\bullet} G \circ F$, so daß für alle c aus $\mathbf{C}$ $\langle c, \eta_c: c \to G(F(c)) \rangle$ ein universeller Morphismus ist.*



Die natürliche Tansformation η wird Einheit, *F der* links adjungierte Funktor, *oder* links Adjunkt *und G der* rechts adjungierte Funktor *oder* rechts Adjunkt *der Adjunktion genannt.*

Eine andere Bezeichung für den links adjungierten Funktor F ist: F ist freier Funktor zu G. Die Bedeutung des links adjungierten Funktors für die Semantik parametrisierten Spezifikationen liegt darin, daß er zu jedem Objekt aus $\mathbf{C}$ einen universellen Morphismus konstruiert.

Definition 1.12 (Persistenz) *Ein links adjungierter Funktor einer Adjunktion $\langle F, G, \eta \rangle: \mathbf{C} \to \mathbf{D}$ ist persistent, wenn η ein natürlicher Isomorphismus ist, d.h. wenn η_c ein Isomorphismus für $c \in \mathbf{C}$ ist.*

Ist η_c die Identität auf c für alle c aus $\mathbf{C}$, dann heißt F streng persistent und es gilt

$$GF = id_{\mathbf{C}}.$$

Diese Definition der Persistenz orientiert sich an der Definition der Persistenz in [WE85]. Bei [EM85] wird nur gefordert, daß GF zu $Id_{\mathbf{C}}$ isomorph ist. Es gibt allerdings Situationen [WE85], in denen GF isomorph zu $Id_{\mathbf{C}}$ ist, aber nicht bzgl. der Einheit der Adjunktion.

In [EM85] ist für Vergißfunktoren zu injektiven Spezifikationsmorphismen in der Gleichungslogik gezeigt, daß es zu jedem persistenten Funktor einen isomorphen streng persistenten Funktor gibt.

Kapitel 2

Institutionen

In [GB90] wird von Goguen und Burstall der Begriff Institution vorgeschlagen um im Bereich der Spezifikation von Software von den, zur Spezifikation verwendeten, Logiken zu abstrahieren. Ziel ist es Eigenschaften und Konstruktionen von Spezifikationen und Spezifikationssprachen so allgemein wie möglich zu formulieren, damit nicht für jede konkrete Logik ähnliche oder gleiche Eigenschaften immer wieder neu bewiesen werden müssen.

Eine Institution besteht

- aus einer Menge von Signaturen, die die abstrakte Syntax einer Logik darstellen,
- für jede Signatur aus einer Menge von Strukturen, die die Menge aller möglichen Interpretationen der abstrakten Syntax festlegen,
- für jede Signatur aus einer Menge von Formeln über den Symbolen der Signatur,
- und für jede Signatur einer Erfüllbarkeitsrelation, die festlegt, wann eine Formel unter einer Interpretation der Symbole der Signatur, d.h. in einer Struktur, gültig ist.

Die einzige Forderung an Institutionen ist, daß bei der Übersetzung der Symbole einer Signatur in Symbole einer anderen Signatur und den entsprechenden Übersetzung der Interpretationen die Gültigkeit von Formeln erhalten bleiben muß.

Definition 2.1 (Institution) *Eine Institution* $I = \langle \mathbf{Sign}, \mathbf{Mod}, \mathbf{Sen}, \models \rangle$ *besteht aus*

- *einer Kategorie* **Sign** *von* Signaturen.
- *einem Funktor*[1] $Mod\colon \mathbf{Sign} \to \mathbf{Cat}^{\mathbf{op}}$, *der Signaturen auf eine Kategorie von* Strukturen *abbildet.*
- *einem Funktor*[2] $Sen\colon \mathbf{Sign} \to \mathbf{Set}$, *der eine Signatur auf eine Menge von* Formeln *abbildet.*
- *und einer Familie von Relationen* $\models = (\models_\Sigma \subseteq |Mod(\Sigma)| \times Sen(\Sigma))_{\Sigma \in \mathbf{Sign}}$.

Erfüllbarkeitsbedingung: Für alle Signaturmorphismen $\sigma\colon \Sigma \to \Sigma'$, *Strukturen* $m' \in Mod(\Sigma')$ *und Formeln* $e \in Sen(\Sigma)$ *muß gelten*

$$m'|_\sigma \models_\Sigma e \qquad gdw. \qquad m' \models_{\Sigma'} \sigma(e).$$

[1] für $Mod(\sigma)(m)$ wird im folgenden $m|_\sigma$ geschrieben, wobei σ ein Morphismus aus **Sign** ist.
[2] für $Sen(\sigma)(e)$ wird im folgenden $\sigma(e)$ geschrieben, wobei σ ein Morphismus aus **Sign** ist.

Eine Formel e ist gültig in einer Struktur m wenn $m \models_\Sigma e$ gilt. Ist Σ aus dem Zusammenhang klar wird auch $m \models e$ statt $m \models_\Sigma e$ geschrieben.

Logiken, die in den Rahmen von Institutionen passen sind z.B. Gleichungslogik, Prädikatenlogik 1. Stufe mit und ohne Gleichheit, Hornklauselogik und Gleichungslogik mit bedingten Gleichungen [GB90].

2.1 Gleichungslogik

Als Beispiel für eine Institution wird gezeigt, daß die Gleichungslogik eine Institution ist [GB90].

Definition 2.2 (Signatur der Gleichungslogik)
Eine Signatur der Gleichungslogik $\Sigma = \langle S, F \rangle$ besteht aus einer endlichen Menge von Sorten S und einer S^+ indizierten Familie $F = (F_{us})_{u \in S^+, s \in S}$ von Funktionssymbolen.[3] *Für $f \in F_{us}$ wird auch $f: u \to s$ geschrieben.*

Zu einer Signatur der Gleichungslogik $\langle S, F \rangle$ sind die Funktionen *sorts* und fun definiert, die die Menge der Sorten S bzw. die Familie der Funktionssymbole F zurückliefern.

Definition 2.3 (Signaturmorphismus) *Ein Signaturmorphismus $\sigma: \Sigma \to \Sigma'$ ist ein Paar $\langle g, h \rangle$, wobei $g: sorts(\Sigma) \to sorts(\Sigma')$ die Sorten von Σ auf Sorten von Σ' abbildet und $h = (h_{us}: fun(\Sigma)_{us} \to fun(\Sigma')_{g^*(us)})_{us \in S^+}$*[4] *eine S^+ indizierte Menge von Funktionen ist, die die Funktionssymbole aus Σ auf Funktionssymbole aus Σ' abbildet.*

Signaturen zusammen mit Signaturmorphismen bilden die Kategorie **GLSign**. Die Identität ist die Identität auf Sorten und Funktionssymbolen und die Komposition von Signaturmorphismen läßt sich auf die Komposition der Funktion für Sorten und der für Funktionssymbole zurückführen.

Definition 2.4 (Σ–Algebra) *Eine Σ–Algebra A besteht aus einer Familie von nicht leeren Mengen $(A_s)_{s \in sorts(\Sigma)}$, den Trägermengen, und zu jedem Funktionssymbol $f \in fun(\Sigma)_{us}$ einer Funktion $f^A: A^u \to A_s$.*[5]

Definition 2.5 (Σ–Homomorphismen) *Für zwei Σ–Algebren A, B ist ein Σ–Homomorphismus $h: A \to B$ eine $S := sorts(\Sigma)$ indizierte Familie von Abbildungen $(h_s: A_s \to B_s)_{s \in S}$, die folgende Homomorphiebedingung erfüllen. Ist f ein Funktionssymbol aus $fun(\Sigma)_{us}$ und $\vec{a} \in A^u$, dann muß gelten:*[6]

$$h(f^A(\vec{a})) = f^B(h^*(\vec{a}))$$

Σ–Algebren und Σ–Homomorphismen bilden eine Kategorie. Der Identitätsmorphismus ist die Familie der Identitäten auf den Trägermengen und die Komposition ist die Familie der Kompositionen der einzelnen Funktionen.

Definition 2.6 (Funktor Alg) *Der Funktor Alg: **GLSign** $\to$ **Cat**$^{\mathbf{op}}$ bildet Signaturen der Gleichungslogik Σ auf die Kategorie ihrer Σ–Algebren ab und Signaturmorphismen $\sigma: \Sigma \to \Sigma'$ auf Funktoren $Mod(\sigma): Mod(\Sigma') \to Mod(\Sigma)$, die wie folgt definiert sind:*

[3] S^* steht für die Menge aller Folgen aus Elementen von S einschließlich der leeren Folge ϵ. S^+ steht für S^* ohne ϵ.

[4] g^* ist die natürliche Erweiterung von g auf Zeichenketten, d.h. $g^*(s_1 \dots s_n) = g(s_1) \dots g(s_n)$.

[5] $A^{s_1 \dots s_n}$ steht für $A_{s_1} \times \dots \times A_{s_n}$

[6] $h^*(\vec{a})$ steht für $h(a_1), \dots, h(a_n)$, falls $\vec{a} = a_1, \dots, a_n$ ist.

- $(A|_\sigma)_s := A_{\sigma(s)}$ *für* $s \in sorts(\Sigma)$ *und* $A \in Mod(\Sigma')$
- $f^{A|_\sigma} := \sigma(f)^A$ *für* $f \in fun(\Sigma)$ *und* $A \in Mod(\Sigma')$
- $(h\colon A \to B)|_\sigma := (h_{\sigma(s)}\colon (A|_\sigma)_s \to (B|_\sigma)_s)_{s \in sorts(\Sigma)}$ *für* $h \in arr(Mod(\Sigma))$ *und* $A, B \in Mod(\Sigma')$

Definition 2.7 (Terme) *Gegeben sei eine* $sorts(\Sigma)$ *indizierte Familie von abzählbaren Mengen* $V = (V_s)_{s \in sorts(\Sigma)}$. *Die Terme der Sorte* s *über der Signatur* Σ *(*$s \in sorts(\Sigma)$*) mit Variablen* V *, i.Z.* $T_\Sigma(V)_s$*, ist induktiv wie folgt definiert:*

- *ist* $v \in V_s$*, dann ist* $v \in T_\Sigma(V)_s$
- *ist* $t_1, \ldots, t_n \in T_\Sigma(V)^{s_1 \ldots s_n}$ *und* $f \in fun(\Sigma)_{us}$ *für* $s, s_i \in sorts(\Sigma)$ *(*$i \in \{1 \ldots n\}$*), dann ist* $f(t_1, \ldots, t_n) \in T_\Sigma(V)_s$.
- *nichts sonst sind Terme.*

$T_\Sigma(V)$ *ist dann die* $sorts(\Sigma)$ *indizierte Familie von Termen* $(T_\Sigma(V)_s)_{s \in sorts(\Sigma)}$.

Die Terme $T_\Sigma(V)$ bilden eine Σ–Algebra A. Die Trägermengen sind die Terme der jeweiligen Sorte: $\forall s \in sorts(\Sigma) : A_s := T_\Sigma(V)_s$. Und $f^A := f$ für alle $f \in fun(\Sigma)_u$ $(u \in sorts(\Sigma)^+)$. Die Terme $T_\Sigma(\emptyset)$ über eine leere Variablenmenge werden als Grundterme T_Σ bezeichnet.

Definition 2.8 (Gleichung) *Eine* Σ*–Gleichung, oder einfach Gleichung, ist ein ungeordnetes Paar* $\{t, t'\}$ *von Termen der gleichen Sorte, d.h.* $t, t' \in T_\Sigma(V)_s$ *für* $s \in sorts(\Sigma)$. *Für Gleichungen* $\{t, t'\}$ *schreibt man auch* $t = t'$.

Der Funktor $GLSen$: **GLSign** $\to$ **Set** bildet Signaturen der Gleichungslogik Σ auf die Menge ihrer Σ–Gleichungen $(GLSen(\Sigma))$ ab und Signaturmorphismen $\sigma\colon \Sigma \to \Sigma'$ auf Funktionen $GLSen(\sigma)$ von $GLSen(\Sigma)$ nach $GLSen(\Sigma')$ (im folgenden ebenfalls mit σ bezeichnet), die wie folgt definiert sind:

- $\sigma(v) := v \in V_{\sigma(s)}$ für $v \in V_s$, wobei σ auf den Variablen injektiv ist.
- $\sigma(f(t_1, \ldots, t_n)) := \sigma(f)(\sigma(t_1), \ldots, \sigma(t_n))$ für $f \in fun(\Sigma)_{s_1 \ldots s_{n+1}}$
- $\sigma(t = t') := \sigma(t) = \sigma(t')$

Definition 2.9 (Variablenbelegung) *Sei* Σ *eine Signatur der Gleichungslogik und* $V = (V_s)$ *eine* $S := sorts(\Sigma)$ *indizierte Familie von Variablen. Eine Variablenbelegung* $b\colon V \to A$ *ist dann eine* S *indizierte Familie von Funktionen* $(b_s\colon V_s \to A_s)_{s \in S}$*, die jeder Variablen* v *aus* V_s *einen Wert* a *aus* A_s *zuordnet.*

Eine Variablenbelegung b läßt sich eindeutig zu einem Homomorphismus $b^*\colon T_\Sigma(V) \to A$ wie folgt erweitern:

- $b^*_s(v) := b_s(v)$ für $v \in V_s$
- $b^*_{s_{n+1}}(f(t_1, \ldots, t_n)) := f^A(b^*_{s_1}(t_1), \ldots, b^*_{s_n}(t_n))$ für $f \in fun(\Sigma)_{s_1 \ldots s_{n+1}}$

Definition 2.10 (Gültigkeit von Gleichungen) *Ist* A *eine* Σ*–Algebra und* $t = t'$ *eine* Σ*–Gleichung (*t *und* t' *aus* $T_\Sigma(V)$*), dann ist* $t = t'$ *in* A *gültig, i.Z.* $A \models_\Sigma t = t'$*, gdw. für alle Variablenbelegungen* $b\colon V \to A$ *gilt:*

$$b^*(t) = b^*(t')$$

In der Algebra, deren Trägermengen nur ein Element haben und deren Funktionen die konstante Funktion sind, sind alle Gleichungen gültig. Diese Algebra wird auch triviale Algebra genannt.

Lemma 2.11 *Gegeben seien ein Signaturmorphismus $\sigma: \Sigma \to \Sigma'$, eine Σ'–Algebra A und Variablenbelegungen $b_A: V \to A$ und $b_{A|_\sigma}: V' \to A|_\sigma$ mit $b_{A|_\sigma}(v') = b_A(\sigma(v'))$. Für alle Terme t aus $T_\Sigma(V)$ gilt dann*

$$b^*_{A|_\sigma}(t) = b^*_A(\sigma(t)).$$

Beweis Der Beweis wird induktiv über den Termaufbau geführt.

1. $b^*_{A|_\sigma}(v') = b_{A|_\sigma}(v') = b_A(\sigma(v')) = b^*_A(\sigma(v'))$ für $v' \in V'$.
2. für $f \in fun(\Sigma)_{s_1 \ldots s_{n+1}}$ gilt dann:

$$\begin{aligned} b^*_{A|_\sigma}(f(t_1, \ldots, t_n)) &= f^{A|_\sigma}(b^*_{A|_\sigma}(t_1), \ldots, b^*_{A|_\sigma}(t_n)) \\ &= \sigma(f)^A(b^*_A(\sigma(t_1)), \ldots, b^*_A(\sigma(t_n))) \\ &= b^*_A(\sigma(f(t_1, \ldots, t_n))) \end{aligned}$$

■

Satz 2.12 (Erfüllbarkeitsbedingung) *Für alle Signaturmorphismen $\sigma: \Sigma \to \Sigma'$, alle Σ'–Algebren und alle Gleichungen $t = t'$ aus $GLSen(\Sigma)$ gilt*

$$A \models \sigma(t) = \sigma(t') \Leftrightarrow A|_\sigma \models t = t'.$$

Beweis

($\Longrightarrow$) Für jede Variablenbelegung $b_{A|_\sigma}: V \to A|_\sigma$ ist

$$b^*_{A|_\sigma}(t) = b^*_{A|_\sigma}(t')$$

zu zeigen. Die Variablenbelegung $b_A: \sigma(V) \to A$ wird definiert als $b_A(\sigma(v)) := b_{A|_\sigma}(v)$. Mit Lemma 2.11 und der Voraussetzung, daß $A \models \sigma(t) = \sigma(t')$ gilt, erhält man

$$b^*_{A|_\sigma}(t) = b^*_A(\sigma(t)) = b^*_A(\sigma(t')) = b^*_{A|_\sigma}(t').$$

($\Longleftarrow$) Die Rückrichtung verläuft analog. Zu einer Variablenbelegung $b_A: V' \to A$ wird $b_{A|_\sigma}: V \to A|_\sigma$ definiert mit $b_{A|_\sigma}(v) := b_A(\sigma(v))$ und man erhält analog zum 1. Teil

$$b^*_A(\sigma(t)) = b^*_{A|_\sigma}(t) = b^*_{A|_\sigma}(t') = b^*_A(\sigma(t')).$$

■

Aus den vorangegangenen Betrachtungen erhält man.

Korollar 2.13 (Institution $\mathcal{GL}$ der Gleichungslogik)
$\mathcal{GL} := \langle \mathbf{GLSign}, \mathbf{Alg}, \mathbf{GLSen}, \models \rangle$ *ist eine Institution.*

2.2 Theorien und Modelle

Dieser Abschnitt beschäftigt sich mit den Strukturen einer Signatur, in denen alle Formeln einer Formelmenge gültig sind, den Modellen und mit der Menge der Formeln einer Signatur, die in einer Menge von Strukturen gültig sind, den Theorien und der Beziehung zwischen beiden. Grundlage für diesen Abschnitt sind [GB90] und [EGL89].

Zuerst kommen einige Abkürzungen, die im folgenden benötigt werden:

- $m \models E \iff \forall e \in E : m \models e$.
- $M \models e \iff \forall m \in M : m \models e$.

Definition 2.14 (Modelle) *Die Modelle $Mod(E)$ einer Menge von Formeln E aus $Sen(\Sigma)$ sind die Strukturen m aus $Mod(\Sigma)$, in denen die Formeln in E gültig sind*

$$Mod(E) := \{m \in Mod(\Sigma) \mid m \models E\}.$$

Definition 2.15 (Theorie) *Die Theorie $Th(M)$ einer Menge von Strukturen M ist die Formelnmenge E, die die Formeln enthält, die in allen Strukturen aus M gültig sind*

$$Th(M) := \{e \in Sen(\Sigma) \mid M \models e\}.$$

Zu einer Menge E von Formeln aus $Sen(\Sigma)$ bezeichnet E^* die Menge von Formeln, die in allen Modellen von E gültig sind

$$E^* := Th(Mod(E)).$$

Die Menge E^* enthält alle logischen Konsequenzen von E. Eine Formelmenge E heißt Theorie, wenn $E = E^*$ ist.

Zu einer Menge von Strukturen aus $Mod(\Sigma)$ bezeichnet M^* die Menge von Strukturen, in denen die Theorie von M gültig ist

$$M^* := Mod(Th(M)).$$

Die Menge M^* wird auch als Abschluß von M bezeichnet. Eine Menge von Strukturen M ist abgeschloßen, wenn $M = M^*$ ist.

Eine Formelmenge E *impliziert* eine Formel e (i.Z. $E \models e$) gdw. e in allen Modellen von E gültig ist ($Mod(E) \models e$). Eine Formelmenge E' *folgt* aus einer Formelmenge E (i.Z. $E \models E'$), gdw. alle Formeln aus E' in den Modellen von E gültig sind ($Mod(E) \models E'$).

Es folgen nun einige einfache Aussagen, die später benötig werden. Die Beweise dazu sind einfach und können z.B. in [EGL89] gefunden werden.

Fakt 2.16 *Sind M und M' Mengen von Strukturen aus $Mod(\Sigma)$ und E und E' Formelmengen mit Formeln aus $Sen(\Sigma)$, dann gilt:*

1. $E \subseteq E' \Rightarrow Mod(E') \subseteq Mod(E) \Rightarrow E \subseteq E'^*$
2. $M \subseteq M' \Rightarrow Th(M') \subseteq Th(M) \Rightarrow M \subseteq M'^*$
3. $E \subseteq E^*$
4. $M \subseteq M^*$

5. $E^* = E^{**}$
6. $M^* = M^{**}$
7. *(Monotonie)* $E \subseteq E' \Rightarrow E^* \subseteq E'^*$
8. *(Monotonie)* $M \subseteq M' \Rightarrow M^* \subseteq M'^*$
9. $Mod(E) = Mod(E') \Longleftrightarrow E^* = E'^*$
10. $Th(M) = Th(M') \Longleftrightarrow M^* = M'^*$

2.3 Formeln mit freien Variablen

In diesem Abschnitt wird ein Konzept für Formeln mit freien Variablen auf Institutionsebene vorgestellt. Im Gegensatz zu freien Variablen einer Logik wird hier der Begriff der offenen Formeln unabhängig von einer konkreten Logik definiert. Die Definitionen und Sätze stammen im wesentlichen aus [ST85b]. Die Idee ist, freie Variablen einer Signatur Σ als zusätzliche Symbole zu Σ in einer neuen Signatur Σ' aufzufassen. In der Gleichungslogik kann man z.B. Variablen als neue Konstantensymbole einer, um diese Symbole erweiterten, Signatur sehen.

Das Σ' eine Erweiterung von Σ ist, läßt sich durch einen Signaturmorphismus $\phi: \Sigma \to \Sigma'$ ausdrücken. Formeln der Signatur Σ mit freien Variablen aus „$\Sigma' - \phi(\Sigma)$“ sind dann einfach Formeln aus $Sen(\Sigma')$.

Definition 2.17 (Offene Formeln) *Ein Paar $\langle \phi, e \rangle$, wobei $\phi: \Sigma \to \Sigma'$ ein Signaturmorphismus und e eine Formel aus $Sen(\Sigma')$ ist, ist eine* offene Σ–Formel *mit freien Variablen „$\Sigma' - \phi(\Sigma)$“.*

Für eine Struktur $m \in Mod(\Sigma)$ ist eine Belegung der zusätzlichen Symbole aus Σ' mit Werten aus m (Belegung der freien Variablen mit Werten aus m) eine Struktur $m' \in Mod(\Sigma')$, bei der das Redukt auf Σ die Struktur m ist.

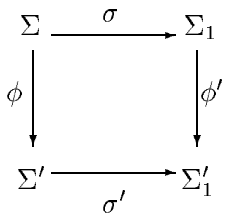
Definition 2.18 (Erfüllbarkeit offener Formeln)
Ist m eine Struktur aus $Mod(\Sigma)$ und $\langle \phi, e \rangle$ eine offene Σ–Formel, dann erfüllt *m mit der Variablenbelegung $m' \in Mod(\Sigma')$ die Formel $\langle \phi, e \rangle$ (i.Z. $m \models_{m'} e$) genau dann wenn e in m' gültig ist.*

Es ist kein Problem entsprechend der Prädikatenlogik 1. Stufe auf der Institutionsebene die Negation, Konjunktion und Allquantifizierung einzuführen. Es ist demnach möglich aus einer Grundinstitution eine komplexe Institution aufzubauen, deren Sätze aus den Verknüpfungen der Sätze der Grundinstitution bestehen [ST85b, Tar84c].

Definition 2.19 (Übersetzung offener Formeln) *Ist $\langle \phi, e \rangle$ eine Σ–Formel ($\phi: \Sigma \to \Sigma'$) und $\sigma: \Sigma \to \Sigma_1$ ein Signaturmorphismus, dann ist $\sigma(\langle \phi, e \rangle)$ definiert als*

$$\sigma(\langle \phi, e \rangle) := \langle \phi', \phi'(e) \rangle,$$

wobei



ein Pushout in **Sign** *ist.*

Die Definition der Übersetzung von offenen Formeln verlangt, daß die Kategorie **Sign** immer Pushouts besitzt. Diese Forderung ist keine große Einschränkung, da z.B. bei parametrisierten Spezifikationen ebenfalls die Kombination von Signaturen gebraucht wird. In [EGL89] z.B. findet man, daß in der Institution $\mathcal{GL}$ der Gleichungslogik die Kategorie **GLSign** Pushouts besitzt.

Es gibt allerdings das Problem [ST85b], daß der Pushout nur bis auf Isomorphie eindeutig bestimmt ist und somit die Übersetzung der offenen Formeln nicht wohldefiniert ist. Glücklicherweise gilt, da Mod und Sen Funktoren sind, daß, wenn $\iota: \Sigma' \to \Sigma''$ ein Isomorphismus in **Sign** mit Inversem ι^{-1} ist, dann sind $Sen(\iota)$ und $Mod(\iota)$ Isomorphismen in **Set** bzw. $\mathbf{Cat}^{\mathbf{op}}$ und wegen der Erfüllbarkeitsbedingung in Institutionen ist für jede Struktur m aus $Mod(\Sigma')$ und jede Formel e aus $Sen(\Sigma')$

$$m \models e \Leftrightarrow (m|_{\iota^{-1}})|_{\iota} \models e \Leftrightarrow m|_{\iota^{-1}} \models \iota(e).$$

Da also die Pushouts bezüglich der Erfüllbarkeit nicht zu unterscheiden sind, reicht es für die Zwecke dieser Arbeit aus, bei der Übersetzung einer offenen Formel einen beliebigen Pushout zu nehmen.

In manchen Fällen will man die Klasse der möglichen Signaturmorphismen, mit denen man offene Formeln bilden kann, einschränken. In der Gleichungslogik z.B. können Variablen als neue Konstantensymbole verstanden werden, neue Funktionssymoble (also Funktionsvariablen) sind i.a. nicht üblich. Die Klasse I der eingeschränkten Signaturmophismen muß dann allerdings gegen die Pushoutbildung in **Sign** abgeschloßen sein. Das heißt, ist $\phi: \Sigma \to \Sigma'$ in I und $\sigma: \Sigma \to \Sigma_1$ ein Signaturmorphismus, dann gibt es einen Pushout in **Sign** so daß der Morphismus $\phi': \Sigma_1 \to \Sigma_1'$ in I ist.

Satz 2.20 (Erfüllbarkeitsbedingung offener Formeln)
Für alle Signaturmorphismen σ von Σ nach Σ_1 aus **Sign**, *alle Strukturen m aus $Mod(\Sigma_1)$ und alle offenen Σ–Formeln e gilt die* Erfüllbarkeitsbedingung für offenen Formeln

$$m \models_{m'} \sigma(e) \Leftrightarrow m|_{\sigma} \models_{m'|_{\sigma'}} e.$$

Beweis Ist $e = \langle \phi, e' \rangle$ eine offene Σ–Formel, dann ist $\sigma(e) = \langle \phi', \sigma'(e') \rangle$ erfüllbar in m mit der Variablenbelegung m'

$$m \models_{m'} \sigma(e)$$

gdw. $\sigma'(e')$ in m' gültig ist

$$m' \models \sigma'(e').$$

Wegen der Erfüllbarkeitsbedingung in Institutionen gilt, daß $m' \models \sigma'(e')$ gdw. e' in $m'|_{\sigma'}$ gültig ist

$$m'|_{\sigma'} \models e'.$$

Das ist nach Definition Erfüllbarkeit genau dann der Fall, wenn e in $m|_\sigma$ mit der Variablenbelegung $m'|_{\sigma'}$ erfüllbar ist

$$m|_\sigma \models_{m'|_{\sigma'}} e.$$

■

Teil II

Parametrisierte Spezifikationen

Kapitel 3

Abstrakte Datentypen und Spezifikationen

3.1 Abstrakte Datentypen

Ein Datentyp ist ein Paar $\langle \Sigma, m \rangle$, wobei $\Sigma \in \mathbf{Sign}$ die Menge von Symbolen bezeichnet, die der Benutzer des Datentyps verwenden kann und die Struktur $m \in Mod(\Sigma)$ eine Realisierung der Symbole aus Σ ist. Ein abstrakter Datentyp abstrahiert von einer konkreten Realisierung einer Signatur, die ein Datentyp darstellt, auf eine Menge von Realisierungen mit gemeinsamen Eigenschaften.

Definition 3.1 (abstrakter Datentyp) *Ein* abstrakter Datentyp $adt = \langle \Sigma, \mathbf{M} \rangle$ *besteht aus*

- *einer Signatur* $\Sigma \in \mathbf{Sign}$
- *und einer Kategorie von Modellen* $\mathbf{M}$*, einer vollen Unterkategorie von* $Mod(\Sigma)$*.*

Die Kategorie **Adt** *der abstrakten Datentypen hat als Objekte abstrakte Datentypen* $\langle \Sigma, \mathbf{M} \rangle$ *und als Morphismen* $\sigma\colon \langle \Sigma, \mathbf{M} \rangle \to \langle \boldsymbol{\Sigma}', \mathbf{M}' \rangle$ *Signaturmorphismen* $\sigma\colon \Sigma \to \Sigma'$*, bei denen* $\mathbf{M}'|_\sigma \subseteq \mathbf{M}$ *ist.*

Daß die Modelle eines abstrakten Datentyps eine Kategorie bilden (s.a. [EGL89]) hat den Vorteil, daß man die initialen Objekte der Modelle eines abstrakten Datentyps oder einen Funktor auf den Modellen abstrakter Datentypen betrachten kann. Da aber von der Modellkategorie gefordert wird, daß sie eine volle Unterkategorie von $Mod(\Sigma)$ sind (s.a. [EGL89]), könnte man die Modelle eines abstrakten Datentyps als Teilmenge der Objekte von $Mod(\Sigma)$ wie z.B. in [Wir89, ST85b] ansehen und nur wenn über Funktoren oder initiale Objekte gesprochen wird sich die Morphismen aus $Mod(\Sigma)$ ausleihen.

Die Kategorie **Adt** für die Institution der Gleichungslogik entspricht der Kategorie **ADT** in [EGL89] bzw. der Kategorie der Spezifikationen **Spec** in [Wir89]. In [ST85b] wird die Kategorie der Spezifikationen **Spec** aus [Wir89] auch für beliebige Institutionen definiert.

Auf abstrakten Datentypen sind zwei Funktionen definiert, $Sign$ und Mod, die die Signatur und die Modelle eines abstrakten Datentyps liefern.

Definition 3.2 (Funktor Sign) *Der Funktor* $Sign\colon \mathbf{Adt} \to \mathbf{Sign}$ *bildet abstrakte Datentypen auf ihre Signaturen und* **Adt***–Morphismen auf ihre Signaturmorphismen ab:*

$$\begin{aligned} Sign(\langle \Sigma, \mathbf{M} \rangle) &:= \Sigma \\ Sign(\sigma\colon adt \to adt') &:= \sigma\colon Sign(adt) \to Sign(adt'). \end{aligned}$$

Definition 3.3 (Funktion Mod)
Zu einem abstrakten Datentyp $adt = \langle \Sigma, \mathbf{M} \rangle$ *liefert* $Mod(adt)$ *die Kategorie der Modelle des abstrakten Datentyps*

$$Mod(\langle \Sigma, \mathbf{M} \rangle) := \mathbf{M}.$$

Operationen auf abstrakten Datentypen

Operationen auf abstrakten Datentypen erzeugen aus vorhandenen abstrakten Datentypen neue. Im folgenden werden nur die für diese Arbeit wichtigsten Operationen aus [ST85b, Wir89] definiert.

Definition 3.4 (Vereinigung) *Sind* $adt_1 = \langle \Sigma, \mathbf{M_1} \rangle$ *und* $adt_2 = \langle \Sigma, \mathbf{M_2} \rangle$ *zwei abstrakte Datentypen gleicher Signatur, dann ist die* Vereinigung *von* adt_1 *und* adt_2 *durch die Schnittmenge ihrer Modelle definiert:*

$$adt_1 \cup adt_2 := \langle \Sigma, \mathbf{M_1} \cap \mathbf{M_2} \rangle.$$

Die Übersetzung mittels eines Signaturmorphismus dient dazu die Signatur eines abstrakten Datentyps um neue Symbole zu erweitern. Als Modelle der Übersetzung erhält man alle Strukturen der Zielsignatur, die die Bedeutung der Symbole der Ausgangssignatur erhalten.

Definition 3.5 (Übersetzung)
Ist $adt = \langle \Sigma, \mathbf{M} \rangle$ *ein abstrakter Datentyp aus* **Adt** *und* $\sigma: \Sigma \to \Sigma'$ *ein Signaturmorphismus, dann ist die* Übersetzung *von* adt *durch* σ *wie folgt definiert*

$$\begin{aligned} Sign(\sigma(adt)) &:= \Sigma' \\ |Mod(\sigma(adt))| &:= \{m \in |Mod(\Sigma')| \mid m|_\sigma \in |\mathbf{M}|\} \\ arr(Mod(\sigma(adt))) &:= \{h \in arr(Mod(\Sigma')) \mid h|_\sigma \in arr(\mathbf{M})\}. \end{aligned}$$

Fakt 3.6 *Ist* $Mod(adt)$ *eine volle Unterkategorie von* $Mod(\Sigma)$*, dann ist* $Mod(\sigma(adt))$ *eine volle Unterkategorie von* $Mod(\Sigma')$*.*

Beweis Sind m und m' aus $Mod(\sigma(adt))$ und ist $h: m \to m'$ ein Morphismus aus $Mod(\Sigma')$, dann ist zu zeigen, daß h auch in $Mod(\sigma(adt))$ liegt. Da m und m' in $Mod(\sigma(adt))$ liegen sind $m|_\sigma$ und $m'|_\sigma$ in $Mod(adt)$. Der Morphismus $h|_\sigma$ ist dann ebenfalls in $Mod(adt)$, da $Mod(adt)$ eine volle Unterkategorie von $Mod(\Sigma)$ ist. Wegen der Definition von $\sigma(adt)$ ist h in $Mod(\sigma(adt))$.
■

Die Ableitung eines abstrakten Datentyps bzgl. eines Signaturmorphismus dient dazu Symbole einer Signatur zu verstecken. Die Strukturen bleiben bei der Ableitung erhalten, allerdings sind Teile der Strukturen nicht mehr zugreifbar.

Definition 3.7 (Ableitung) *Ist* $\sigma: \Sigma \to \Sigma'$ *ein Signaturmorphismus, und* $adt' = \langle \Sigma', \mathbf{M}' \rangle$ *ein abstrakter Datentyp, dann ist der abstrakte Datentyp* $adt'|_\sigma$ *definiert durch:*

$$\begin{aligned} Sign(adt'|_\sigma) &= \Sigma \\ \textit{und } Mod(adt'|_\sigma) &= \mathbf{M}'|_\sigma. \end{aligned}$$

Die Ableitung entspricht der Einschränkung der Implementierung eines Moduls auf sein Exportinterface.

Ein Abstraktor α ist eine Operation auf abstrakten Datentypen, die von den einzelnen Modellen eines abstrakten Datentyps abstrahiert. Dem Abstraktor liegt eine Äquivalenzrelation $\equiv_\alpha$ auf $|Mod(Sign(adt))|$ zugrunde und $Beh_\alpha(adt)$ enthält als Modelle alle zu einem Modell aus adt $\equiv_\alpha$–äquivalenten Strukturen aus $Mod(Sign(adt))$. Die Definition eines Abstraktors stammt aus [ST86].

Definition 3.8 (Abstraktor) *Ein Abstraktor α, bestimmt durch eine Äquivalenzrelation $\equiv_\alpha \subseteq |Mod(\Sigma)| \times |Mod(\Sigma)|$, ist eine Operation, die abstrakte Datentypen adt mit Signatur Σ auf deren Abstraktion bzgl. α abbilden*

$$Beh_\alpha(adt) := \langle \Sigma, \equiv_\alpha(Mod(adt)) \rangle.$$

Beispiele für Abstraktoren sind *obs* und *beob*, die durch die Verhaltensäquivalenz bzw. die Äquivalenz bzgl. beobachtbarer Sorten bestimmt sind (s.a. Abschnitte 8.1 und 8.2. Aber auch die Operation **ISOCLOSE** aus [ST85b] oder [EGL89], die den Abschluß der Modelle eines abstrakten Datentyps unter Isomorphie bildet, ist ein Abstraktor, bei dem zwei Strukturen äquivalent sind, wenn sie isomorph sind.

Abstraktoren α für abstrakte Datentypen einer Signatur Σ lassen sich mithilfe eines Signaturmorphismus $\sigma: \Sigma \to \Sigma'$ auf Abstraktoren für abstrakte Datentypen der Signatur Σ' übersetzen, indem $\equiv_\alpha$ mithilfe von σ auf eine Äquivalenzrelation auf $|Mod(\Sigma')|$ abgebildet wird.

Definition 3.9 (Übersetzung von Relationen) *Ist $\sigma: \Sigma \to \Sigma'$ ein Signaturmorphismus und $\equiv$ eine Relation auf $|Mod(\Sigma)|$, dann ist für Strukturen m und m' aus $Mod(\Sigma')$*

$$m \; \sigma(\equiv) \; m'$$

gdw.

$$m|_\sigma \equiv m'|_\sigma.$$

Definition 3.10 (Übersetzung von Abstraktoren) *Ist α ein Abstraktor für abstrakte Datentypen der Signatur Σ und $\sigma: \Sigma \to \Sigma'$ ein Signaturmorphismus, dann ist $\sigma(\alpha)$ ein Abstraktor für abstrakte Datentypen der Signatur Σ', wobei die Äquivalenzrelation $\equiv_{\sigma(\alpha)}$ als $\sigma(\equiv_\alpha)$ definiert ist.*

Es ist leicht zu zeigen, daß wenn $\equiv_\alpha$ eine Äquivalenzrelation ist, $\sigma(\equiv_\alpha)$ ebenfalls eine Äquivalenzrelation ist. Damit ist sichergestellt, daß $\sigma(\alpha)$ ein Abstraktor ist.

Für die Behandlung der Stabilität (s.a. Abschnitt 10) wird die Ableitung von Relationen bzgl. eines Signaturmorphismus benötigt.

Definition 3.11 (Ableitung von Relationen) *Für einen Signaturmorphismus $\sigma: \Sigma \to \Sigma'$ und Relation $\equiv \subseteq |Mod(\Sigma')| \times |Mod(\Sigma')|$ ist die Relation $\equiv|_\sigma$ definiert als $\{\langle m|_\sigma, m'|_\sigma \rangle \mid \langle m, m' \rangle \in \equiv\}$.*

Im allgemeinen ist $\equiv|_\sigma$ keine Äquivalenzrelation mehr, wenn $\equiv$ eine Äquivalenzrelation ist, da $\equiv|_\sigma$ nicht transitiv sein muß wie folgendes Beispiel zeigt.

Beispiel Wähle m_1,m_2,m_3 und m_4 aus $Mod(\Sigma')$ mit $m_1 \equiv m_2$, $m_3 \equiv m_4$, m_1 und m_4 nicht $\equiv$–äquivalent, und $m_2|_\sigma = m_3|_\sigma$. Dann gilt, daß $m_1|_\sigma$ und $m_2|_\sigma$ $\equiv|_\sigma$–äquivalent und $m_2|_\sigma = m_3|_\sigma$ und $m_4|_\sigma$ $\equiv|_\sigma$–äquivalent sind, aber nicht mehr $m_1|_\sigma$ und $m_4|_\sigma$.

Aus diesen einfachen Operationen können komplexere, wie z.B. die Anwendung parametrisierter abstrakter Datentypen auf abstrakte Datentypen, gebildet werden. Mithilfe von parametrisierten abstrakten Datentypen ist es dann möglich sich eigene Operationen auf abstrakten Datentypen zu definieren.

3.2 Spezifikationen

Eine Spezifikation im Sinne dieser Arbeit (s.a. [EM85, EGL89]) bestimmt einen abstrakten Datentyp. Eine Spezifikation gibt die Signatur eines abstrakten Datentyps an und die Eigenschaften, ausgedrückt mithilfe von Formeln aus $Sen(\Sigma)$, die zur Festlegung der Modelle des abstrakten Datentyps dienen.

Definition 3.12 (Spezifikation) *Eine Spezifikation* $spec := \langle \Sigma, E \rangle$ *besteht aus*

- *einer Signatur* $\Sigma \in$ **Sign**
- *und einer Menge von Formeln* $E \subseteq Sen(\Sigma)$

Die Kategorie **Spec** *der Spezifikationen hat als Objekte Spezifikationen* $\langle \Sigma, E \rangle$ *und als Morphismen* $\sigma: \langle \Sigma, E \rangle \to \langle \Sigma', E' \rangle$ *Signaturmorphismen* $\sigma: \Sigma \to \Sigma'$*, mit der Einschränkung, daß* $E' \models \sigma(E)$ *gilt, d.h. die Formeln in* $\sigma(E)$ *aus denen in* E' *folgen.*

Die Definition der Kategorie **Spec** für die Institution der Gleichungslogik entspricht der Definition der Kategorie SPEC in [EM85] und [EGL89].

Wie bei abstrakten Datentypen liefern zwei Funktoren, $Sign$ und Sen, die Bestandteile einer Spezifikation.

Definition 3.13 (Funktor Sign) *Der Funktor* $Sign$: **Spec** $\to$ **Sign** *bildet Spezifikationen auf ihre Signaturen und* **Spec***–Morphismen auf ihre Signatur–Morphismen ab:*

$$Sign(\langle \Sigma, E \rangle) := \Sigma$$

$$Sign(\sigma: spec \to spec') := \sigma: Sign(spec) \to Sign(spec')$$

Definition 3.14 (Funktor Sen) *Der Funktor* Sen: **Spec** $\to$ **Set** *bildet Spezifikationen auf die Menge ihrer Formeln ab.*

$$Sen(\langle \Sigma, E \rangle) := E$$

$$Sen(\sigma: spec \to spec') := Sen(Sign(\sigma))$$

Zu einer Spezifikation ist die Menge von Strukturen assoziiert, in denen die Formeln der Spezifikation gültig sind. Diese Menge ist Basis für die lose und initiale Semantik von Spezifikationen.

Definition 3.15 (Funktion Mod**)** *Auf den Objekten der Kategorie* **Spec** *ist die Funktion* Mod *definiert, die zu einer Spezifikation* $spec = \langle \Sigma, E \rangle$ *die volle Unterkategorie von* $Mod(\Sigma)$ *liefert, die alle Strukturen* m *enthält, in denen die Formeln in* E *gültig sind*

$$|Mod(\langle \Sigma, E \rangle)| := Mod(E).$$

Operationen auf Spezifikationen

Ähnlich wie bei abstrakten Datentypen kann man auch auf Spezifikationen Operationen definieren. Die wichtigsten für diese Arbeit sind die Vereinigung und die Übersetzung mittels eines Signaturmorphismus, weil man mit ihnen die Anwendung parametrisierter Spezifikationen auf Spezifikationen beschreiben kann.

Definition 3.16 (Vereinigung) *Sind* $spec_1 = \langle \Sigma, E_1 \rangle$ *und* $spec_2 = \langle \Sigma, E_2 \rangle$ *zwei Spezifikationen mit gleicher Signatur, dann ist die* Vereinigung *von* $spec_1$ *und* $spec_2$ *durch die Vereinigung der Formelmengen* E_1 *und* E_2 *definiert*

$$spec_1 \cup spec_2 := \langle \Sigma, E_1 \cup E_2 \rangle.$$

Definition 3.17 (Übersetzung) *Ist* $spec = \langle \Sigma, E \rangle$ *eine Spezifikation und* $\sigma: \Sigma \to \Sigma'$ *ein Signaturmorphismus, dann ist die* Übersetzung *von spec durch* σ *bestimmt durch die Übersetzung der Formeln in* E *mittels* $Sen(\sigma)$

$$\sigma(spec) := \langle \Sigma', \sigma(E) \rangle.$$

Der Zusammenhang zwischen den Operationen auf Spezifikationen und den entsprechenden Operationen auf abstrakten Datentypen wird im nächsten Abschnitt behandelt.

3.3 Semantik von Spezifikationen

Eine Spezifikation $spec = \langle \Sigma, E \rangle$ bestimmt einen abstrakten Datentypen $adt = \langle \Sigma, \mathbf{M} \rangle$ der gleichen Signatur. Die Bedeutung einer Spezifikation kann man als eine Funktion von Objekten der Kategorie **Spec** auf Objekte der Kategorie **Adt** auffassen.

Definition 3.18 (Semantik von *spec*) *Eine* Semantik *von Spezifikationen ist eine Funktion, die Spezifikationen auf abstrakte Datentypen der gleichen Signatur abbildet*

$$sem: |\mathbf{Spec}| \to |\mathbf{Adt}|,$$

wobei $Sign(spec) = Sign(sem(spec))$ *ist.*

Die Semantikfunktion *sem* ist i.a. kein Funktor, da sich die Spezifikationsmorphismen nicht bei jeder Semantik auf **Adt**–Morphismen abbilden lassen. Dieser Fall tritt z.B. bei der initialen Semantik auf, wenn die Bedingungen „no–junk“ und „no–confusion“ von dem Spezifikationsmorphismus verletzt werden (s.a. Seite 59).

Die natürlichste Semantik einer Spezfikation ist die lose Semantik („loose semantics“ z.B. [ST85b]), die einer Spezifikation den abstrakten Datentyp zuordnet, der als Modelle alle die Strukturen enthält, in denen die Formeln der Spezifikation gültig sind.

Definition 3.19 (lose Semantik) *Ist spec eine Spezifikation, dann ist die* lose Semantik *von spec definiert als*

$$loose(spec) := \langle Sign(spec), Mod(spec) \rangle.$$

Die lose Semantik hat den Nachteil, daß man um z.B. eine Modellmenge zu spezifizieren, die nur isomorphe Strukturen enthält, eine stärkere Logik als die Gleichungslogik braucht. Das liegt daran, daß in der trivialen Algebra (s.a. Seite 18) alle Gleichungen gültig sind und damit die

triviale Algebra in der losen Semantik jeder Spezifikation enthalten ist. Das ist u.a. deswegen der Fall, weil $GLSen(\Sigma)$ keine Ungleichungen enthält. Eine Ungleichung der Form $true \neq false$ z.B. würde die trivale Algebra ausschließen, da die Trägermenge der Sorte von $true$ und $false$ mindestens zwei Elemente haben müsste.

Eine Lösung des Problems ist es, nicht mehr alle Strukturen $Mod(spec)$ als Modelle der Semantik von *spec* zuzulassen. Eine Einschränkung z.B. ist, nur die initialen Modelle der Kategorie $Mod(spec)$ für die Modelle der Semantik zu wählen. Die initialen Modelle haben den Vorteil, daß sie bis auf Isomorphie eindeutig bestimmt sind und in ihnen nur die Gleichheiten auf Grundtermen aus T_Σ gültig sind, die in allen Strukturen aus $Mod(spec)$ gültig sind.

Definition 3.20 (initiale Semantik) *Ist* $spec = \langle \Sigma, E \rangle$ *eine Spezifikation, dann ist die* initiale Semantik *von spec definiert als:*

$$init(\langle \Sigma, E \rangle) := \langle \Sigma, \mathbf{M} \rangle.$$

Die Kategorie $\mathbf{M}$ *ist die volle Unterkategorie von* $Mod(\Sigma)$*, welche die initialen Objekte von* $Mod(spec)$ *enthält.*

In der Gleichungslogik gilt, daß die initialen Modelle von $Mod(spec)$ immer existieren und isomorph zu $T_\Sigma/\equiv_E$ sind. In Logiken, die über die Hornklausellogik hinausgehen, gibt es i.a. keine initialen Modelle mehr.[1]

Wie sieht es mit der Verträglichkeit der initialen und losen Semantik mit den in Abschnitt 3.2 definierten Operationen auf Spezifikationen aus? Das heißt, gilt

$$\begin{aligned} sem(spec_1 \cup spec_2) &= sem(spec_1) \cup sem(spec_2) \\ \text{und } sem(\sigma(spec)) &= \sigma(sem(spec)) \end{aligned}$$

für sem = loose bzw. sem = init? Die folgenden zwei Sätze zeigen die Verträglichkeit der losen Semantik mit der Vereinigung und Übersetzung von Spezifikationen.

Satz 3.21 (Verträglichkeit der Vereinigung mit der losen Semantik)
Sind $spec_1 = \langle \Sigma, E_1 \rangle$ *und* $spec_2 = \langle \Sigma, E_2 \rangle$ *Spezifikationen, dann gilt:*

$$loose(spec_1 \cup spec_2) = loose(spec_1) \cup loose(spec_2).$$

Beweis Es reicht zu zeigen, daß $Mod(spec_1 \cup spec_2)$ gleich $Mod(spec_1) \cap Mod(spec_2)$ ist. Sei also m eine Struktur aus $Mod(spec_1 \cup spec_2)$, dann ist m aus $Mod(\langle \Sigma, E_1 \cup E_2 \rangle)$. Die Struktur m erfüllt somit die Formeln in E_1 und in E_2 und damit ist m in $Mod(E_1)$ und in $Mod(E_2)$, also in $Mod(spec_1) \cap Mod(spec_2)$.

Sei nun m eine Struktur aus $Mod(spec_1) \cap Mod(spec_2)$, dann ist m in $Mod(E_1)$ und in $Mod(E_2)$ und erfüllt somit die Formeln in $E_1 \cup E_2$. Damit ist m aber auch in $Mod(spec_1 \cup spec_2)$.

Die Klasse der Morphismen von $Mod(spec_1 \cup spec_2)$ und $Mod(spec_1) \cap Mod(spec_2)$ sind gleich, da beide, wegen der Definition von Mod und $\cap$, volle Unterkategorien von $Mod(\Sigma)$ sind und die gleichen Objekte haben. ■

Satz 3.22 (Verträglichkeit der Übersetzung mit der losen Semantik)
Ist $spec = \langle \Sigma, E \rangle$ *eine Spezifikation und* $\sigma: \Sigma \to \Sigma'$ *ein Signaturmorphismus, dann gilt:*

$$loose(\sigma(spec)) = \sigma(loose(spec)).$$

[1] s.a. die Arbeiten von Tarlecki über Institutionen, die initiale Semantik erlauben. [Tar84a, Tar84b, Tar84c]

Beweis Es ist zu zeigen, daß $Mod(\sigma(spec))$ gleich der Menge aller Strukturen m' aus $Mod(\Sigma')$ ist, für die das Redukt von m' auf Σ in $Mod(spec)$ liegt

$$Mod(\sigma(spec)) = \{m' \in Mod(\Sigma') \mid m'|_\sigma \in Mod(spec)\}.$$

Sei m' aus $Mod(\sigma(spec))$, dann ist $m'|_\sigma$ auch in $Mod(spec)$, da m' die Formeln in $\sigma(E)$ erfüllt und wegen der Erfüllbarkeitsbedingung in Institutionen auch $m'|_\sigma$ die Formeln in E erfüllt.

Umgekehrt, ist m' aus $\sigma(Mod(spec))$, dann ist zu zeigen, daß m' in $Mod(\sigma(spec))$ liegt, also die Formeln in $\sigma(E)$ erfüllt. Das gilt ebenfalls wegen der Erfüllbarkeitsbedingung und der Tatsache, daß $m'|_\sigma$ in $Mod(spec)$ liegt und damit die Formeln in E erfüllt.

Nach Fakt 3.6 ist $\sigma(Mod(spec))$ eine volle Unterkategorie von $Mod(\Sigma')$ und damit sind die Morphismen in $\sigma(Mod(spec))$ und $Mod(\sigma(spec))$ gleich. ■

Die initiale Semantik dagegen ist nicht verträglich mit der Vereinigung und Übersetzung von Spezifikationen.

Beispiel Als Gegenbeispiel für die Vereinigung betrachte die Signatur Σ aus **GLSign**, die eine Sorte s und zwei Konstanten c und k der Sorte s hat. Die Signatur der Spezifikationen $spec_1$ und $spec_2$ ist Σ und die Formelmenge von $spec_1$ leer und die von $spec_2$ enthält die Gleichung $k = c$. Die Trägermenge der Sorte s der Modelle in $init(spec_1)$ muß zweielementig und die der Modelle von $init(spec_2)$, wegen der Gleichung $k = c$, einelementig sein. Die Schnittmenge der Modelle von $init(spec_1)$ und $init(spec_2)$ ist damit leer. Andererseits ist $Mod(init(spec_1 \cup spec_2))$ nicht leer, da in diesem Fall $spec_1 \cup spec_2$ gleich $spec_2$ ist.

Beispiel Für die Übersetzung betrachte die Signatur Σ mit der Sorte s, der Konstanten c und die Signatur Σ' mit der gleichen Sorte s und Konstanten c wie in Σ plus einer zusätzlichen Konstante k der Sorte s. Als Signaturmorphismus $\sigma: \Sigma \to \Sigma'$ wähle die Inklusion und als Spezifikation wähle $spec = \langle \Sigma, \emptyset \rangle$. Die Trägermenge der Sorte der Modelle von $init(spec)$ sind einelementig, die der Modelle von $init(\sigma(spec))$ dagegen zweielementig, wegen der Existenz der zweiten Konstante k der Sorte s. Ist A eine Σ'–Algebra und liegt $A|_\sigma$ in $Mod(init(spec))$, d.h. A liegt in $Mod(\sigma(init(spec)))$, dann kann die Trägermenge der Sorte s von A wegen der Definition des Redukts $A|_\sigma$, ebenfalls nur einelementig sein. Folglich können $init(\sigma(spec))$ und $\sigma(init(spec))$ nicht gleich sein.

Kapitel 4

Parametrisierung

4.1 Parametrisierte abstrakte Datentypen

Ein parametrisierter abstrakter Datentyp besteht aus einem abstrakten Datentyp als formalem Parameter, einem abstrakten Datentyp als Rumpf und einen **Adt**–Morphismus zwischen ihnen. Die Modelle des Rumpfes sind Erweiterungen der Modelle des formalen Parameters um neue Symbole und deren Bedeutung. Diese Erweiterungen müssen nicht eindeutig sein, noch muß für jede formale Parameterstruktur eine Erweiterung existieren. Gibt es zu jeder Parameterstruktur eine eindeutige Erweiterung, dann definiert ein parametrisierter abstrakter Datentypen einen Funktor (s.a. Abschnitt 5.1).

Definition 4.1 (parametrisierter abstrakter Datentyp)
Ein parametrisierte abstrakter Datentyp $B(X\colon adt_F)_\beta\colon adt_B$ *besteht aus einem abstrakten Datentyp* adt_F *als formalem Parameter, einem abstrakten Datentyp* adt_B *als Rumpf und einem* **Adt***–Morphismus* β *zwischen formalem Parameter und Rumpf.*

$$B(X\colon adt_F)_\beta\colon adt_B := \beta\colon adt_F \to adt_B \in \mathbf{Adt}$$

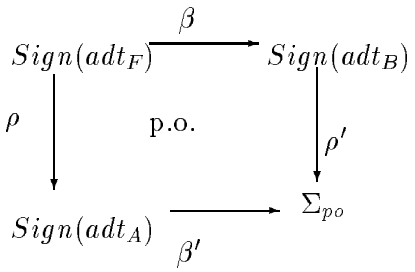
Ein abstrakter Datentyp adt_A ist zulässiger aktueller Parameter eines parametrisierten abstrakten Datentyps, wenn seine Signatur alle Symbole der Signatur des formalen Parmeters enthält (e.v. umbenannt) und wenn die aktuellen Parametermodelle Erweiterungen von formalen Parametermodellen sind. Die Redukte der Modelle des aktuellen Parameters müssen Modelle des formaeln Parameters sein, da nur für diese der Rumpf eine mögliche Erweiterung bereitstellt.

Definition 4.2 (Zulässiger aktueller Parameter) *Ein abstrakter Datentyp* adt_A *ist zulässiger aktueller Parameter für einen parametrisierten abstrakten Datentyp* $B(X\colon adt_F)_\beta\colon adt_B$ *via* ρ*, gdw.* $\rho\colon adt_F \to adt_A$ *ein* **Adt***–Morphismus ist.*

Wenn im folgenden geschrieben wird, daß adt_A ein zulässiger aktueller Parameter für B ist, wird die Existenz eines geeigneten **Adt**–Morphismus von adt_F nach adt_A vorausgesetzt und mit ρ (ρ, ρ_1, ρ' etc.) bezeichnet.

Das Ergebnis der Anwendung eines parametrisierten abstrakten Datentyps B auf einen abstrakten Datentyp adt_A ergibt wieder einen abstrakten Datentyp $B(adt_A)$. Die Signatur von $B(adt_A)$ besteht aus den Symbolen der Signaturen des Rumpfes und des aktuellen Parameters, wobei darauf geachtet wird, daß Symbole der aktuellen Parametersignatur, die nicht in der formalen Parametersignatur enthalten sind, nicht mit den Symbolen der Rumpfsignatur

zufällig kollidieren. Diese Bedingung erfüllt das Pushoutobjekt des folgenden Diagramms in der Kategorie **Sign**.



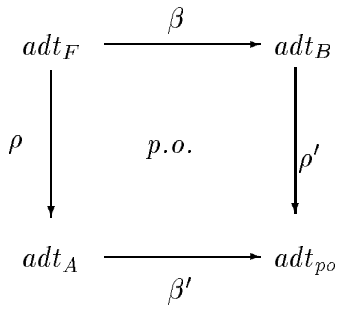
In der Gleichungslogik existieren immer Pushouts in **GLSign** [EM85, EGL89] und damit Pushouts in **Adt** [EGL89]. Sind $\beta\colon \Sigma_F \to \Sigma_A$ und $\rho\colon \Sigma_F \to \Sigma_A$ Inklusionen, erhält man als Σ_{po}:

$$\Sigma_{po} = \langle S_F \cup (S_B - S_F) \cup (S_A - S_F), F_F \cup (F_B - F_F) \cup (F_A - F_F) \rangle.$$

In diesem Fall ist die diskjunkte Vereinigung gemeint. Sind ρ und β keine Inklusionen, daß muß man die Umbenennung von Sorten und Funktionen des Parameters berücksichtigen.

Die Modelle von $B(adt_A)$ sind dann alle Strukturen aus den $Mod(\Sigma_{po})$, die Erweiterungen von aktuellen Parameter– und Rumpfmodellen sind.

Definition 4.3 (Anwendung parametrisierter abstrakter Datentypen)
Ist $B(X\colon adt_F)_\beta\colon adt_B$ ein parametrisierter abstrakter Datentyp und adt_A ein zulässiger aktueller Parameter für B, dann ist die Anwendung von B auf adt_A ($B(adt_A)$) als das Pushoutobjekt folgenden Diagramms in **Adt** *definiert*

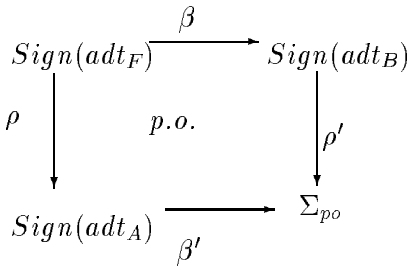


und es gilt:

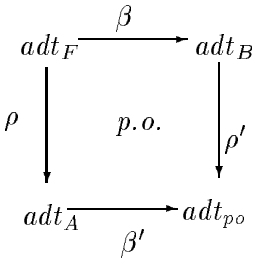
$$B(adt_A) := adt_{po} = \beta'(adt_A) \cup \rho'(adt_B).$$

Die Frage bleibt, wann der Pushout in **Adt** und damit die Anwendung von B auf adt_A definiert ist. Der folgende Satz sagt aus, daß dies immer dann der Fall ist, wenn man die Resultatsignatur bilden kann, d.h. die Signaturen einen Pushout in **Sign** bilden.

Satz 4.4 (Der Funktor Sign reflektiert Pushouts) *Ist $D = adt_A \xleftarrow{\rho} adt_F \xrightarrow{\beta} adt_B$ ein Diagramm in* **Adt** *und existiert zu dem Diagramm $Sign(D) = Sign(adt_A) \xleftarrow{\rho} Sign(adt_F) \xrightarrow{\beta} Sign(adt_B)$ in* **Sign** *ein Pushout,*



dann hat auch D einen Pushout mit $Sign(adt_{po}) = \Sigma_{po}$.



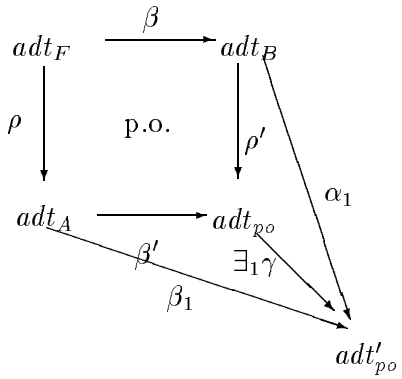
Die Kategorie $Mod(adt_{po})$ ist dann bis auf Isomorphie eindeutig durch die Strukturen und Morphismen aus $Mod(\Sigma_{po})$ bestimmt, die in $Mod(adt_B)$ und in $Mod(adt_A)$ liegen

$$adt_{po} = \beta'(adt_A) \cup \rho'(adt_B).$$

Beweis Für die Pushouteigenschaft reicht es zu zeigen, daß die Morphismen $\beta'\colon Sign(adt_A) \to \Sigma_{po}$ und $\rho'\colon adt_B \to adt_{po}$ **Adt**–Morphismen sind und, daß der Pushout universell in **Adt** ist.

Um zu zeigen, daß ρ' und β' **Adt**–Morphismen sind muß $Mod(adt_{po})|_{\beta'} \subseteq Mod(adt_A)$ und $Mod(adt_{po})|_{\rho'} \subseteq Mod(adt_B)$ sein. Ist $m \in Mod(adt_{po})$, dann ist nach Definition der Vereinigung von abstrakten Datentypen m in $Mod(\beta'(adt_A))$ und $Mod(\rho'(adt_B))$ und nach der Definition der Übersetzung von abstrakten Datentypen sind dann $m|_{\beta'}$ in $Mod(adt_A)$ und $m|_{\rho'}$ in $Mod(adt_B)$.

Für die Universalität des Pushouts nehmen wir an, es gibt eine Erweiterung $E' = adt_A \xrightarrow{\beta_1} adt'_{po} \xleftarrow{\rho_1} adt_B$ des Diagramms D zu einem kommutierenden Rechteck,



dann ist zu zeigen, daß es einen eindeutigen **Adt**–Morphismus $\gamma\colon adt_{po} \to adt'_{po}$ gibt, mit $\gamma\beta' = \beta_1$ und $\gamma\rho' = \rho_1$. Wegen der Pushouteingeschaft von Σ_{po} gibt es einen eindeutigen Signaturmorphismus $\gamma\colon Sign(adt_{po}) \to \Sigma'_{po}$ mit $\gamma\beta' = \beta_1$ und $\gamma\rho' = \rho_1$.

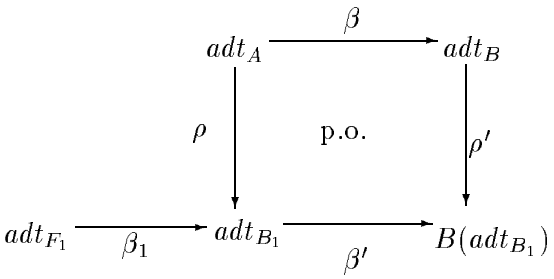
Es bleibt zu zeigen, daß γ auch ein **Adt**–Morphismus ist, d.h. $Mod(adt'_{po})|_\gamma \subseteq Mod(adt_{po})$. Ist $m' \in Mod(adt'_{po})$, dann ist $m'|_{\beta_1} \in Mod(adt_A)$ und $m'|_{\rho_1} \in Mod(adt_B)$. Und damit ist $(m'|_\gamma)|_{\beta'} \in Mod(adt_A)$ und $(m'|_\gamma)|_{\rho'} \in Mod(adt_B)$. Wegen der Definition von adt_{po} ist dann $m'|_\gamma \in Mod(adt_{po})$. ■

Die Pushoutkonstruktion erlaubt es auch paramterisierte abstrakte Datentypen B_1 als zulässige aktuelle Parameter zu betrachten. Das Resultat der Anwendung von B auf B_1 ist die Verkettung von B und B_1[1].

Definition 4.5 *Ein parametrisierter abstrakter Datentyp $B_1(X\colon adt_{F_1})_{\beta_1}\colon adt_{B_1}$ ist zulässiger aktueller Parameter für einen parametrisierten abstrakten Datentyp $B(X\colon adt_F)_\beta\colon adt_B$, gdw. der Rumpf adt_{B_1} von B_1 ein zulässiger aktueller Parameter für B ist.*

Die Anwendung von von B auf B_1 ergibt wieder einen parametrisierten abstrakten Datentyp

$$(B_1 \circ B)(X\colon adt_{F_1})_{\beta'\circ\beta_1}\colon B(adt_{B_1}).$$



Da die Kombination von Pushouts wieder Pushouts ergibt (s.a. [Mac88] und [EM85]), ist es egal ob zuerst $B \circ B_1$ berechnet wird und dann das Ergebnis auf einen zulässigen Parameter adt_{A_1} angewandt wird, oder ob zuerst $B_1(adt_{A_1})$ berechnet wird und anschließend B auf das Ergebnis angewandt wird

$$(B \circ B_1)(adt_{A_1}) = B(B_1(adt_{A_1})).$$

4.2 Parametrisierte Spezifikationen

Parametrisierte Spezifikationen sind analog zu parametrisierten abstrakten Datentypen definiert. Statt der Kategorie **Adt** wird die Kategorie **Spec** genommen. Die Definition parametrisierter Spezifikationen entspricht aber auch der in [EM85, EGL89].

Definition 4.6 (parametrisierte Spezifikation)

Eine parametrisierte Spezifikation $B(X\colon spec_F)_\beta\colon spec_B$ besteht aus einer formalen Parameterspezifikation $spec_F$, einer Rumpfspezifikation $spec_B$ und einem **Spec***–Morphismus β zwischen ihnen.*

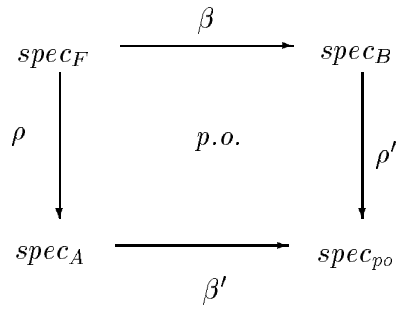
$$B(X\colon spec_F)_\beta\colon spec_B := \beta\colon spec_F \to spec_B \in \mathbf{Spec}$$

Definition 4.7 (Zulässiger Parameter) *Eine Spezifiktion $spec_A$ ist ein zulässiger aktueller Parameter für eine parametrisierten Spezifikation $B(X\colon spec_F)_\beta\colon spec_B$ via ρ, gdw. $\rho\colon spec_F \to spec_A$ ein* **Spec***–Morphismus ist.*

[1] für parametrisierte Spezifikationen s.a. [EM85, EGL89]

Wenn im folgenden geschrieben wird, daß $spec_A$ ein zulässiger aktueller Parameter für B ist, wird die Existenz eines geeigneten **Spec**–Morphismus von $spec_F$ nach $spec_A$ vorausgesetzt und mit ρ (ρ_1, ρ' etc.) bezeichnet.

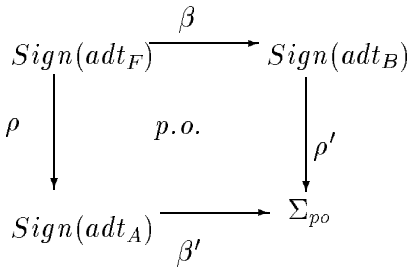
Definition 4.8 (Anwendung parametrisierter Spezifikationen) *Ist $B(X\colon spec_F)_\beta\colon spec_B$ eine parametrisierte Spezifikation und $spec_A$ ein zulässiger aktueller Parameter für B, dann ist die Anwendung von B auf $spec_A$ ($B(spec_A)$) als das Pushoutobjekt folgenden Diagramms definiert*



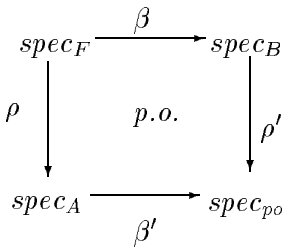
und es ist

$$B(spec_A) := spec_{po} = \beta'(spec_A) \cup \rho'(spec_B).$$

Satz 4.9 (Der Funktor Sign reflektiert pushouts)
Ist $D = spec_A \xleftarrow{\rho} spec_F \xrightarrow{\beta} spec_B$ ein Diagramm in **Spec** *und existiert zu dem Diagramm $Sign(D) = Sign(spec_A) \xleftarrow{\rho} Sign(spec_F) \xrightarrow{\beta} Sign(spec_B)$ in* **Sign** *ein Pushout,*



dann hat auch D einen Pushout mit $Sign(spec_{po}) = \Sigma_{po}$



und es ist

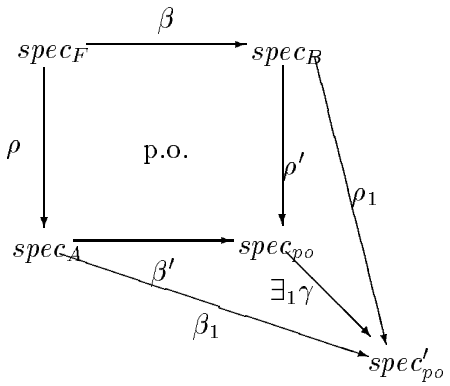
$$spec_{po} = \beta'(spec_A) \cup \rho'(spec_B).$$

Beweis Für die Pushouteigenschaft reicht es zu zeigen, daß die Morphismen $\beta'\colon Sign(spec_A) \to \Sigma_{po}$ und $\rho'\colon spec_B \to spec_{po}$ **Spec**–Morphismen sind und, daß der Pushout universell in **Spec** ist.

Um zu zeigen, daß ρ' und β' **Spec**–Morphismen sind muß $Sen(spec_{po}) \models \beta'(Sen(spec_A))$ und $Sen(spec_{po}) \models \rho'(Sen(spec_A))$ gelten. Das gilt, da aus der Definition von $spec_{po}$ folgt:

$$Sen(spec_{po}) = \beta'(Sen(spec_A)) \cup \rho'(Sen(spec_B)).$$

Für die Universalität des Pushouts nehmen wir an, es gibt eine Erweiterung $E' = spec_A \xrightarrow{\beta_1} spec'_{po} \xleftarrow{\rho_1} spec_B$ des Diagramms D zu einem kommutierenden Rechteck,



dann ist zu zeigen, daß es einen eindeutigen **Spec**–Morphismus $\gamma: spec_{po} \to spec'_{po}$ gibt, mit $\gamma\beta' = \beta_1$ und $\gamma\rho' = \rho_1$. Wegen der Pushouteigenschaft von Σ_{po} gibt es einen eindeutigen Signaturmorphismus $\gamma: Sign(spec_{po}) \to \Sigma'_{po}$ mit $\gamma\beta' = \beta_1$ und $\gamma\rho' = \rho_1$.

Es bleibt zu zeigen, daß γ auch ein **Spec**–Morphismus ist, d.h.

$$Sen(spec'_{po}) \models \gamma(Sen(spec_{po})).$$

Ist e eine Formel aus $Sen(spec_{po})$, dann gibt es aufgrund der Definition von $spec_{po}$ entweder ein e_A in $Sen(spec_A)$ mit $e = \beta'(e_A)$ oder ein e_B in $Sen(spec_B)$ mit $e = \rho'(e_B)$. Sei oBdA. $e = \beta'(e_A)$, dann gilt, da $\gamma\beta' = \beta_1$ und β_1 ein **Spec**–Morphismus ist, daß

$$Sen(spec_{po}) \models \beta_1(e_A) = \gamma(e).$$

■

4.3 Semantik parametrisierter Spezifikationen

In [EGL89, EM85] ist die Semantik von parametrisierten Spezifikationen eine Funktor von $Mod(spec_F)$ nach $Mod(spec_B)$. In dieser Arbeit dagegen ist es natürlicher als Semantik einer parametrisierten Spezifikation einen parametrisierten abstrakten Datentyp zu nehmen. In den folgenden Kapiteln wird dann gezeigt unter welchen Bedingungen parametrisierte abstrakte Datentypen einen Funktor definieren und daß die klassischen Aussagen [EM85], wie das Amalgamierungs und das Extension Lemma, gelten.

Eine Semantik einer parametrisierten Spezifikation $B(X: spec_F)_\beta: spec_B$ ist ein parametrisierter abstrakter Datentyp $B(X: adt_F)_\beta: adt_B$, wobei

$$Sign(\beta: adt_F \to adt_B) = Sign(\beta: spec_F \to spec_B)$$

ist.

Man möchte, daß die Pushout Konstruktion mit der Semantik für Spezfikationen und parametrisierten Spezifikationen verträglich ist ist, d.h. für alle $B(X\colon spec_F)_\beta\colon spec_B$ und zulässigen Parameterspezifikationen $spec_A$ soll gelten:

$$sem(B(spec_A)) = sem(B)(sem(spec_A)).$$

Satz 4.10 *Ist $B(X\colon adt_F)_\beta\colon adt_B$ eine Semantik von $B(X\colon spec_F)_\beta\colon spec_B$, seien adt_A und $spec_A$ zulässige aktuelle Parameter und gilt, daß $adt_i \subseteq loose(spec_i)$ für $i \in \{F, B, A\}$ ist, dann ist*

$$B(adt_A) \subseteq loose(B(spec_A))$$

Beweis Sei $spec_i = \langle \Sigma_i, E_i \rangle$ für $i \in \{F, B, A\}$. Es ist zu zeigen, daß jedes $m \in Mod(B(adt_A))$ die Formeln in $E_{po} := Sen(B(spec_A))$ erfüllt.

Sei nun $m \in Mod(B(adt_A))$ und $e \in E_{po}$. Ist $e = \rho(e_B)$, dann erfüllt $m|_{\rho'}\ e_B$, da $m|_{\rho'} \in Mod(adt_B) = Mod(E_B)$ ist. Dann gilt mit der Erfüllbarkeitsbedingung für Institutionen $m \models e$. Analog verläuft der Fall für $e = \beta'(e_A)$. ■

Satz 4.11 *Gilt für $B(X\colon adt_F)_\beta\colon adt_B$ und $B(X\colon spec_F)_\beta\colon spec_B$ mit zulässigem aktuellen Parameter $spec_A$ für B, daß $adt_i = loose(spec_i)$ für $i \in \{F, B\}$, dann ist $loose(spec_A)$ aktueller Parameter für den parametrisierten abstrakten Datentyp B und es gilt*

$$B(loose(spec_A)) = loose(B(spec_A)).$$

Beweis Für $B(adt_A)$ kann man auch

$$B(adt_A) = \beta'(adt_A) \cup \rho'(adt_B)$$

schreiben, und da $adt_i = loose(spec_i)$ (für $i = F, B, A$) ist, erhält man

$$B(adt_A) = \beta'(loose(spec_A)) \cup \rho'(loose(spec_B))$$

Wegen der Verträglichkeit der losen Semantik mit den Operationen auf abstrakten Datentypen erhält man

$$B(adt_A) = loose(\beta'(spec_A) \cup \rho'(spec_B))$$

Die rechte Seite der obigen Gleichung entspricht der Definition von $B(spec_A)$ und damit ist

$$B(adt_A) = loose(B(spec_A)).$$

■

Kapitel 5

Eigenschaften parametrisierter abstrakten Datentypen

5.1 Eindeutigkeit und Konsistenz

Im Gegensatz zu [EM85] wird in dieser Arbeit die Semantik von parametrisierten Spezifikationen als Funktion auf abstrakten Datentypen betrachtet. Im folgenden werden die Bedingungen untersucht, mit denen ein parametrisierter abstrakten Datentyp eine Funktion auf Strukturen beschreibt. Wenn ein parametrisierter abstrakter Datentyp eindeutig und konsistent ist, dann definiert B einen eindeutigen, streng persistenten Funktor von der Kategorie der Modelle des aktuellen Parameters adt_A zu der Kategorie der Modelle von $B(adt_A)$. Die Semantik in [EGL89] einer parametrisierten Spezifikation als Funktor auf Strukturen ist demnach ein Spezialfall der in dieser Arbeit vorgestellten Semantik, bei der die Modelle des Rumpfes geeignet gewählt wurden.[1]

Um die Auswirkungen eines parametrisierten abstrakten Datentyps B auf einzelne Parameterstrukturen und Parametermorphismen zu beschreiben wird die Funktion B_R definiert.

Definition 5.1 (B_R) *Zu einem parametrisierten abstrakten Datentyp B und einem zulässigen aktuellen Parameter adt_A ist eine Funktion B_R: $Mod(adt_A) \to \mathcal{P}(Mod(B(adt_A)))$ definiert, die jeder Struktur m_A aus $Mod(adt_A)$ eine Menge von Strukturen aus $Mod(B(adt_A))$ zuordnet, bei denen das Redukt auf die aktuelle Parametersignatur Σ_A die Struktur m_A ist.*

$$B_R(m_A) := \{m \in Mod(B(adt_A)) \mid m|_{\beta'} = m_A\}$$

Entsprechend ist auf den Morphismen aus $Mod(adt_A)$ die Funktion B_R: $arr(Mod(adt_A)) \to \mathcal{P}(arr(Mod(B(adt_A))))$

$$B_R(h_A) := \{h \in Mod(B(adt_A)) \mid h|_{\beta'} = h_A\}$$

definiert.

Liefert B_R für jede Struktur m_A und jeden Morphismus h_A eines zulässigen Parameters adt_A eine Menge mit mindestens einem Element wird B_R konsistent für adt_A genannt. Die Konsistenz

[1] Für die Wahl der Modelle im Rumpf s.a. Abschnitt 6 über die Verträglichekit der initialen Semantik mit Parametrisierung.

von B bedeutet, daß man eine Menge von Funktionen $\mathcal{F}$ definieren kann, die Strukturen auf Strukturen abbildet:

$$\mathcal{F} := \{F\colon Mod(adt_A) \to Mod(B(adt_A)) \mid F(m_A) \in B_R(m_A)\}.$$

Wegen der Konsistenz von B sind die Funktionen der Menge $\mathcal{F}$ auf jeder Struktur aus $Mod(adt_A)$ definiert.[2]

Definition 5.2 (Konsistenz von B) *Ein parametrisierter abstrakter Datentyp B ist* konsistent *für einen zulässigen aktuellen Parameter adt_A, wenn für jede Struktur m_A und jeden Morphismus h_A aus der Kategorie $Mod(adt_A)$ die Mengen $B_R(m_A)$ und $B_R(h_A)$ mindestens ein Element enthalten.*

Ein parametrisierter abstrakter Datentyp B heißt konsistent, *wenn B für jeden aktuellen Parameter konsistent ist.*

Die Funktionen in $\mathcal{F}$ lassen sich mit dem Begriff von Konsistenz, wie er hier gegeben ist, nicht zu einer Menge von Funktoren erweitern, da die Komposition der Morphismen aus $B_R(h_A)$ und $B_R(h'_A)$ nicht definiert sein muß, auch wenn die Komposition von h_A und h'_A definiert ist.

Es kann sein, daß $B_R(h_A)$ nur $h\colon m \to m_1$ enthält und $B_R(h'_A)$ nur $h'\colon m_2 \to m_3$. Es gilt zwar, daß $m_1|_{\beta'} = m_2|_{\beta'}$ ist, aber es ist nicht $m_1 = m_2$ gewährleistet.

Definition 5.3 (Eindeutigkeit von B) *Ein parametrisierter abstrakter Datentyp B ist* eindeutig *für einen zulässigen aktuellen Parameter adt_A, wenn für jede Struktur m_A und jeden Morphismus h_A aus der Kategorie $Mod(adt_A)$ die Mengen $B_R(m_A)$ und $B_R(h_A)$ höchstens ein Element enthalten.*

Ein parametrisierter abstrakter Datentyp B heißt eindeutig, *wenn B für jeden aktuellen Parameter eindeutig ist.*

Zusammen mit der Konsistenz macht die Eindeutigkeit aus B_R einen Funktor von Strukturen aus $Mod(adt_A)$ auf Strukturen aus $Mod(B(adt_A))$. Der zu einem parametrisierten abstrakten Datentyp B assoziierte Funktor ist dann B_F.

Satz 5.4 *Ist ein parametrisierter abstrakter Datentyp B konsistent und eindeutig für einen aktuellen Parameter adt_A, dann definiert für m_A aus $Mod(adt_A)$*

$$\begin{aligned} B_F(m_A) &:= m \in B_R(m_A) \\ B_F(h_A\colon m_A \to m'_A) &:= h\colon B_F(m_A) \to B_F(m'_A) \in B_R(h_A) \end{aligned}$$

einen streng persistenten, surjektiven Funktor B_F von $Mod(adt_A)$ nach $Mod(B(adt_A))$.

Beweis Es ist erstens zu zeigen, daß B_F eine Funktion auf Objekten und Morphismen der Kategorie $Mod(adt_A)$ ist und zweitens, daß B_F die Funktoreigenschaften erfüllt.

Da B eindeutig und konsistent ist, enthält $B_F(m_A)$ genau ein Element. Damit ist $B_F(m_A)$ wohldefiniert. Ebenfalls enthält $B_R(h_A\colon m_A \to m'_A)$ genau ein Element $h\colon m \to m'$, wobei wegen der Definition von $B_R(h_A)$ die Struktur m in $B_R(m_A)$ und m' in $B_R(m'_A)$ liegt. Also ist $B_F(m_A)$ gleich m und $B_F(m'_A)$ gleich m'. Damit ist dann $B_F(h_A)$ wohldefiniert.

Für die Funktoreigenschaften von B_F betrachte zuerst $B_F(id_{m_A})$. Die Identität $id_{B_F(m_A)}$ ist ein Morphismus aus $Mod(B(adt_A))$, da $B_F(m_A)$ in $Mod(B(adt_A))$ liegt und $Mod(B(adt_A))$

[2] vergleiche auch [SST90]. Die Konsistenzeigenschaft wird dort „excludes local inconsistencies“ genannt.

eine Kategorie ist. Andererseits ist, da $Mod(\beta')$ ein Funktor ist, auch $id_{B_F(m_A)}|_{\beta'} = id_{m_A}$ und damit nach Definition von B_R und B_F das Ergebnis $B_F(id_{m_A})$ gleich $id_{B_F(m_A)}$.

Betrachte nun $B_F(hh')$. Die Komposition $B_F(h) \circ B_F(h')$ ist dann in $Mod(B(adt_A))$, weil erstens $B_F(h)$ und $B_F(h')$ in $Mod(B(adt_A))$ liegen, zweitens die Komposition von $B_F(h)$ und $B_F(h')$ definiert ist, weil hh' definiert ist und drittens $Mod(B(adt_A))$ eine Kategorie ist. Andererseits ist, da $Mod(\beta')$ ein Funktor ist auch $(B(h) \circ B_F(h'))|_{\beta'}$ gleich $B_F(h)|_{\beta'} \circ B_F(h')|_{\beta'}$ gleich hh', und damit $B_F(hh') = B_F(h) \circ B_F(h')$.

Der Funktor B_F ist wegen der Definition von B_R streng persistent. Außerdem ist B_F surjektiv, da für jedes m aus $Mod(B(adt_A))$ das Redukt von m auf $Sign(adt_A)$ in $Mod(adt_A)$ liegen muß, es also ein m_A geben muß für daß $m \in B_R(m_A)$ und damit $m = B_F(m_A)$ ist. ■

Umgekehrt bestimmt die Existenz eines streng persistenten Funktors von $Mod(adt_A)$ nach $Mod(B(adt_A))$, daß der parametrisierte abstrakte Datentyp B konsistent für adt_A ist. Ist der Funktor zusätzlich noch surjektiv, dann ist B eindeutig für adt_A.

Satz 5.5 *Sei B ein parametrisierter abstrakter Datentyp und adt_A zulässiger aktueller Parameter von B. Gibt es einen streng persistenten Funktor $F\colon Mod(adt_A) \to Mod(B(adt_A))$ bzgl. $Mod(\beta')\colon Mod(B(adt_A)) \to Mod(adt_A)$, dann ist B konsistent für adt_A.*

Ist F außerdem surjektiv, dann ist B eindeutig für adt_A.

Beweis Zuerst wird gezeigt, daß B konsistent ist, d.h. für alle m_A aus $Mod(adt_A)$ gibt es ein m aus $Mod(B(adt_A))$ mit $m|_{\beta'} = m_A$. Definiere m als $F(m_A)$, dann ist m in $Mod(B(adt_A))$ und wegen der strengen Persistenz von F, ist $m|_{\beta'} = F(m_A)|_{\beta'} = m_A$. Analog zeigt man, daß $F(h_A)$ in $B_R(h_A)$ liegt.

Ist F zusätzlich surjektiv, dann ist zu zeigen, daß B eindeutig ist, d.h. $B_R(m_A)$ maximal ein Element für m_A aus $Mod(adt_A)$ enthält. Seien m und m' aus $B_R(m_A)$, dann existieren, wegen der Surjektivität von F, zwei Strukturen m'_A und m''_A mit $F(m'_A) = m$ und $F(m''_A) = m'$. Da F streng persistent ist, gilt daß $m|_{\beta'} = F(m'_A)|_{\beta'} = m'_A$ und $m'|_{\beta'} = F(m''_A)|_{\beta'} = m''_A$ ist.

Andererseits liegen m und m' in $B_R(m_A)$ und damit muß $m|_{\beta'} = m_A = m'|_{\beta'}$ sein. Es ergibt sich also, daß $m_A = m'_A = m''_A$ und damit $m = F(m'_A) = F(m''_A)$ ist. Entsprechend wird gezeigt, daß $B_R(h_A)$ maximal ein Element enthält. ■

Die Definitionen der Konsistenz und Eindeutigkeit entsprechen denen bei [Sch86] für Zellen. Eine Zelle die konsistent und eindeutig ist, wird dort auch Modul genannt.

Aus den letzten beiden Sätzen 5.5 und 5.4 folgt, daß wenn B einen Funktor beschreibt, dieser auch eindeutig ist.

Korollar 5.6 *Ein parametrisierter abstrakter Datentyp B ist eindeutig und konsistent für adt_A, gdw. es einen streng persistenten, surjektiven Funktor $F\colon Mod(adt_A) \to Mod(B(adt_A))$ gibt. In diesem Fall sind B_F und F gleich.*

Beweis Es ist nur noch zu zeigen, daß B_F gleich F ist. Seien $B_F(m_A)$ und $F(m_A)$ für ein m_A aus $Mod(adt_A)$ verschieden, dann gilt aber wegen der strengen Persistentz von F daß $F(m_A)|_{\beta'} = m_A$ ist.

Außerdem ist $F(m_A)$ in $Mod(B(adt_A))$. Nach Definition von B_R ist dann aber $F(m_A)$ in $B_R(m_A)$. Da B eindeutig und konsistent ist, enthält B_R genau das Element $B_F(m_A)$. Also ist $F(m_A) = B_F(m_A)$, was im Widerspruch zu der Annahme, daß $F(m_A)$ und $B_F(m_A)$ ungleich sind, steht. Also ist F gleich B_F. ■

5.2 Amalgamierte Summe und Extension Lemma

Bei der Anwendung eines parametrisierten abstrakten Datentyps B auf einen abstrakten Datentyp adt_A erhebt sich die Frage, wie sich die Modelle von $B(adt_A)$ aus den Modellen des aktuellen Parameters und des Rumpfes konstruieren lassen. Gibt es überhaupt ein m aus $Mod(B(adt_a))$, so daß $m|_{\beta'} = m_A$ und $m|_{\rho'} = m_B$ ist und ist m dann eindeutig? Ist die Frage mit ja zu beantworten, dann kann man zu den Elementen m_A aus $Mod(adt_A)$ die Strukturen von $B_R(m_A)$ eindeutig aus den Strukturen von $B_R(m_A|_\rho)$ konstruieren.

Definition 5.7 (Amalgamierte Summe) *In einer Institution $\mathcal{I}$* existieren eindeutige amalgamierte Summen, *gdw. es zu jedem Pushout in* **Sign**

$$\begin{array}{ccc} \Sigma_F & \xrightarrow{\beta} & \Sigma_B \\ {\scriptstyle\rho}\downarrow & p.o. & \downarrow{\scriptstyle\rho'} \\ \Sigma_A & \xrightarrow[\beta']{} & \Sigma_{po} \end{array}$$

und je zwei Strukturen $m_A \in Mod(\Sigma_A)$ und $m_B \in Mod(\Sigma_B)$ mit $m_A|_\rho = m_F = m_B|_\beta$ eine eindeutige Struktur $m \in Mod(\Sigma_{po})$ gibt, so daß $m|_{\beta'} = m_A$ und $m|_{\rho'} = m_B$ ist. Und zu je zwei Morphismen $h_A \in Mod(\Sigma_A)$ und $h_B \in Mod(\Sigma_B)$ mit $h_A|_\rho = h_F = h_B|_\beta$ existiert genau ein $h \in Mod(\Sigma_{po})$ mit $h|_{\beta'} = h_A$ und $h|_{\rho'} = h_B$.

Die eindeutigen Konstruktionen von m und h werden amalgamierte Summen *von m_A und m_B via m_F bzw. h_A und h_B via h_F, i.Z.*

$$\begin{aligned} m &= m_A +_{m_F} m_B \\ h &= h_A +_{h_F} h_B, \end{aligned}$$

genannt.

Das Amalgamierungs Lemma besagt, daß die amalgamierte Summe eines aktuellen Parametermodells mit einem Rumpfmodell immer ein Modell von $B(adt_A)$ ist und umgekehrt, daß jedes Modell von $B(adt_A)$ eine amalgamierte Summe eines aktuellen Parametermodells und eines Rumpfmodells ist. Für die Gleichungslogik findet man das Amalgamierungs Lemma z.B. in [EM85].

Satz 5.8 (Amalgamierungs Lemma)
Existieren in der Institution $\mathcal{I}$ eindeutige amalgamierte Summen dann gilt, ist B ein parametrisierter abstrakter Datentyp und adt_A ein zulässiger Parameter für B, dann ist für Strukturen $m_A \in Mod(adt_A)$ und $m_B \in Mod(adt_B)$, für die die amalgamierte Summe definiert ist, $m_A +_{m_F} m_B$ in $Mod(B(adt_A))$.

Entsprechend ist die amalgamierte Summe von Morphismen aus $Mod(adt_A)$ und $Mod(adt_B)$ in $Mod(B(adt_A))$.

Umgekehrt läßt sich jede Struktur und jeder Morphismus aus $Mod(B(adt_A))$ als amalgamierte Summe von Strukturen und Morphismen aus $Mod(adt_A)$ und $Mod(adt_B)$ darstellen.

Beweis Für die amalgamierte Summe $m := m_A +_{m_F} m_B$ gilt, daß $m|_{\beta'}$ gleich m_A in $Mod(adt_A)$ liegt und $m|_{\rho'}$ gleich m_B in $Mod(adt_B)$, also liegt m wegen der Definition von $B(adt_A)$ in $Mod(B(adt_A))$.

Andererseits ist für ein m aus $Mod(B(adt_A))$ das Redukt $m|_{\beta'}$ gleich m_A und $m|_{\rho'}$ gleich m_B. Außerdem ist $m_A|_\rho$ gleich m_F gleich $m_B|_\beta$ und damit erhält man

$$m = m_A +_{m_F} m_B.$$

Entsprechend verläuft der Beweis für die amalgamierte Summe von Morphismen. ■

Lemma 5.9 *Existieren in einer Institution $\mathcal{I}$ eindeutige amalgamierte Summen, dann gelten folgende Rechenregeln für die amalgamierte Summe von Morphismen h_F, h_B, h_A, g_F, g_B und g_A und Strukturen m_F, m_A und m_B:*

1. $id_{m_A} +_{id_{m_F}} id_{m_B} = id_{m_A +_{m_F} m_B}$
2. $h_A g_A +_{h_F g_F} h_B g_B = (h_A +_{h_F} h_B)(g_A +_{g_F} g_B)$.

Beweis

1. Wegen der Funktoreigenschaft von $Mod(\rho')$ und $Mod(\beta')$ erhält man daß

$$\begin{aligned} (id_{m_A +_{m_F} m_B})|_{\beta'} &= id_{m_A} \\ \text{und } (id_{m_A +_{m_F} m_B})|_{\rho'} &= id_{m_B}. \end{aligned}$$

   Wegen der Eindeutigkeit der amalgamierten Summe gilt dann die Gleichheit.

2. Wegen der Funktoreigenschaft von $Mod(\rho')$ und $Mod(\beta')$ erhält man daß

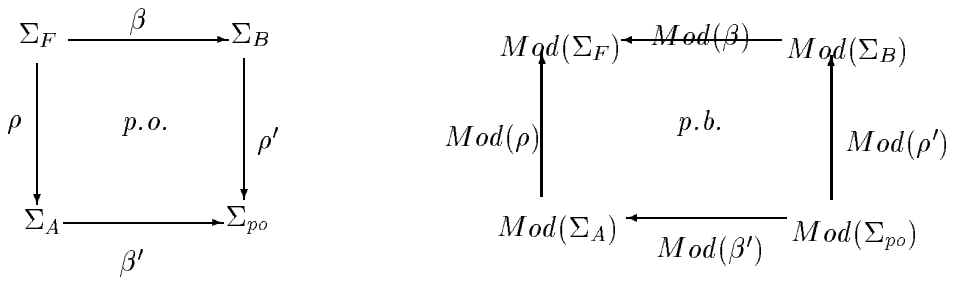
$$\begin{aligned} ((h_A +_{h_F} h_B)(g_A +_{g_F} g_B))|_{\beta'} &= h_A g_A \\ \text{und } ((h_A +_{h_F} h_B)(g_A +_{g_F} g_B))|_{\rho'} &= h_B g_B. \end{aligned}$$

   Wegen der Eindeutigkeit der amalgamierten Summe gilt dann die Gleichheit.

■

Hinreichende und notwendige Bedingung für die eindeutige Existenz amalgamierter Summen in Institutionen $\langle \mathbf{Sign}, \mathbf{Mod}, \mathbf{Sen}, \models \rangle$ ist die Eigenschaft des Funktors Mod Pushouts in **Sign** auf Pullbacks in **Cat** abzubilden.

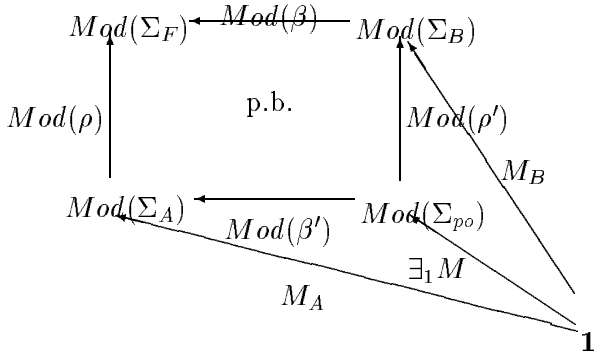
Satz 5.10 *In einer Institution $\mathcal{I} = \langle \mathbf{Sign}, \mathbf{Mod}, \mathbf{Sen}, \models \rangle$ existieren eindeutige amalgamierte Summen gdw. der Funktor Mod Pushouts in* **Sign** *auf Pullbacks in* **Cat** *abbildet.*



Beweis ($\Longleftarrow$)[3] Es ist zu zeigen, daß, wenn $Mod(\Sigma_{po})$ ein Pullback in **Cat** ist, es eindeutige amalgamierte Summen $m = m_A +_{m_F} m_B$ und $h = h_A +_{h_F} h_B$ in $Mod(\Sigma_{po})$ gibt. Für den Beweis der Existenz von m betrachte die Funktoren $M_A: \mathbf{1} \to \mathbf{Mod}(\mathbf{\Sigma_A})$ und $M_B: \mathbf{1} \to \mathbf{Mod}(\mathbf{\Sigma_B})$, bei denen das einzige Objekt in **1** auf m_A bzw. m_B abgebildet wird. Da $m_A|_\rho = m_B|_\beta$ gilt, erhält man

$$Mod(\rho)M_A = Mod(\beta)M_B.$$

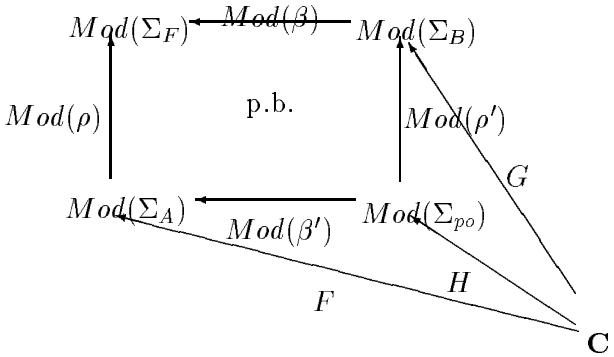
Folglich existiert ein eindeutiger Funktor $M: \mathbf{1} \to \mathbf{Mod}(\mathbf{\Sigma_{po}})$ und damit eine eindeutige Struktur $m = M(1) = m_A +_{m_F} m_B$.



Analog verläuft der Beweis für die amalgamierte Summe von Morphismen $h = h_A +_{h_F} h_B$, wobei statt der Kategorie **1** die Kategorie **2** genommen wird. Die Kategorie **2** enthält genau zwei Objekte 1 und 2 und genau einen Morphismus $g: 1 \to 2$, der nicht die Identität ist. Es werden dann die Funktoren $H_A: \mathbf{2} \to \mathbf{Mod}(\mathbf{\Sigma_A})$ und $H_B: \mathbf{2} \to \mathbf{Mod}(\mathbf{\Sigma_B})$ betrachtet, bei denen g auf h_A bzw. h_B abgebildet wird.

($\Longrightarrow$)[4] Zu zeigen ist, daß zu einer Kategorie **C** und Funktoren $F: \mathbf{C} \to \mathbf{Mod}(\mathbf{adt_A})$ und $G: \mathbf{C} \to \mathbf{Mod}(\mathbf{adt_B})$ für die gilt, daß $Mod(\rho)F = Mod(\beta)G$ ist, ein eindeutiger Funktor $H: \mathbf{C} \to \mathbf{Mod}(\mathbf{\Sigma_{po}})$ existiert mit

$$\begin{aligned} Mod(\beta')H &= F \\ \text{und } Mod(\rho')H &= G. \end{aligned}$$



Definiere für Objekte c und Morphismen h aus **C**

$$\begin{aligned} H(c) &= F(c) +_{F(c)|_\rho} G(c) \\ \text{und } H(h) &= F(h) +_{F(h)|_\rho} G(h). \end{aligned}$$

[3] Diesen Teil des Beweises findet man in [WE85].

[4] Für die Gleichungslogik findet man diesen Teil des Beweises in [EGL89].

Wegen der Eindeutigkeit amalgamierter Summen, ist H wohldefiniert und wegen der Rechenregeln für amalgamierte Summen (Lemma 5.9) auch ein Funktor.

Wegen der Definition der amalgamierten Summe ist

$$\begin{aligned} (Mod(\beta')(H))(c) &= H(c)|_{\beta'} = F(c) \\ \text{und } (Mod(\rho')(H))(c) &= H(c)|_{\rho'} = G(c). \end{aligned}$$

Analog gilt für Morphisem h aus $\mathbf{C}$, daß

$$\begin{aligned} (Mod(\beta')(H))(h) &= H(h)|_{\beta'} = F(h) \\ \text{und } (Mod(\rho')(H))(h) &= H(h)|_{\rho'} = G(h). \end{aligned}$$

ist.

Bleibt zu zeigen, daß H eindeutig ist. Annahme, es gibt einen zweiten Funktor $H'\colon \mathbf{C} \to \mathbf{Mod}(\mathbf{\Sigma_{po}})$ mit $Mod(\beta')H' = F$ und $Mod(\rho')H' = G$, dann ist für $c \in \mathbf{C}$ aber $H'(c)|_{\beta'} = F(c)$ und $H'(c)|_{\rho'} = G(c)$. Wegen der Eindeutigkeit amalgamierter Summen ist dann $H'(c) = H(c)$. Analog gilt für die Morphismen $h \in \mathbf{C}$, daß $H'(h) = H(h)$ ist, und man erhält $H' = H$. ■

Als Folge des vorherigen Satzes hängt die eindeutige Existenz von amalgamierten Summen nur von der Wahl der Kategorie **Sign** und des Funktors Mod ab, aber nicht von der Art der Formeln in $Sen(\Sigma)$ oder der Erfüllbarkeitsrelation $\models$.

Daß es in der Gleichungslogik eindeutige amalgamierte Summen gibt, ist eine bekannte Tatsache [EM85, EGL89].

Die amalgamierte Summe $A = A_A +_{A_F} A_B$ zweier Algebren A_A und A_B, deren Redukte auf der gemeinsamen Signatur gleich sind, ist wie folgt definiert:

$$\begin{aligned} A_s &= \begin{cases} (A_A)_s & \text{falls } s = \beta'(s') \text{ und } s' \in sorts(\Sigma_A) \\ (A_B)_s & \text{falls } s = \rho'(s') \text{ und } s' \in sorts(\Sigma_B) \end{cases} \\ \text{und } f^A &= \begin{cases} f^{A_A} & \text{falls } f = \beta'(f') \text{ und } f' \in fun(\Sigma_A) \\ f^{A_B} & \text{falls } f = \rho'(f') \text{ und } f' \in fun(\Sigma_B) \end{cases} \end{aligned}$$

Die Mengen A_s und und Funktionen f^A sind wohldefiniert, da wegen der Definition des Pushouts, für den Fall daß $\beta'(s') = \rho'(s'')$ bzw. $\beta'(f') = \rho'(f'')$, die Sorten s' und s'' Bild einer Sorte aus Σ_F bzw. die Operationssymbole f und f'' Bild eines Operationssymbols aus Σ_F sein müssen und die Redukte von A_A und A_B auf Σ_F gleich sind.

Bei der Definition der amalgamierten Summe auf Algebren bieten zusätzliche Prädikatensymbole keine Schwierigkeiten, und man erhält, daß auch die Institutionen der Prädikatenlogik 1. Stufe mit und ohne Gleichheit und Hornklausellogik eindeutige amalgamierte Summen besitzen, da diese Eigenschaft unabhängig von den Formeln der Logik und dem Gültigkeitsbegriff ist.

Extension Lemmata

Die amalgamierte Summe erlaubt es in vielen Fällen Eigenschaften, die für den formalen Parameter eines parametrisierten abstrakten Datentyps gelten auf alle zulässigen aktuellen Parameter zu übertragen. Die folgenden Sätze sind Beispiele dafür.

Satz 5.11 (Extension Lemma) *Existieren in der Institution $\mathcal{I}$ eindeutige amalgamierte Summen, dann gilt:*

1. *Ist ein parametrisierter abstrakter Datentyp B konsistent für seinen formalen Parameter adt_F, dann ist B konsistent für alle zulässigen aktuellen Parameter.*

2. *Ist B eindeutig für seinen formalen Parameter, dann ist B eindeutig für alle zulässigen Parameter.*

Beweis (1) Für die Konsistenz ist zu zeigen, daß für alle $m_A \in Mod(adt_A)$ die Menge $B_R(m_A)$ mindestens ein Element enthält. Da B konsistent für adt_F ist, existiert ein $m_B \in B_R(m_A|_\rho)$. Wegen dem Amalgamierungs Lemma (Satz 5.8) ist $m := m_A +_{m_A|_\rho} m_B$ in $Mod(B(adt_A))$, also ist $m \in B_R(m_A)$ und damit $B_R(m_A)$ nicht leer.

Analog zeigt man, daß für einen Morphismus $h_A \in Mod(adt_A)$ die Menge $B_R(h_A)$ nicht leer ist.

(2) Für die Eindeutigkeit ist zu zeigen, daß für alle $m_A \in Mod(adt_A)$ die Menge $B_R(m_A)$ maximal ein Element enthält. Nach dem Amalgamierungs Lemma (Satz 5.8) laßen sich die Modelle von $B(adt_A)$ durch amalgamierte Summen von Morphismen und Strukturen aus $Mod(adt_A)$ und $Mod(adt_B)$ darstellen. Speziell für B_R erhält man:

$$\begin{aligned} B_R(m_A) &= \{m_A +_{m_A|_\rho} m_B \mid m_B \in Mod(adt_B), m_B|_\beta = m_A|_\rho\} \\ B_R(h_A) &= \{h_A +_{h_A|_\rho} h_B \mid h_B \in Mod(adt_B), h_B|_\beta = h_A|_\rho\}. \end{aligned}$$

Die Bedingungen $m_B \in Mod(adt_B)$ und $m_B|_\beta = m_A|_\rho$ bzw. $h_B \in Mod(adt_B)$ und $h_B|_\beta = h_A|_\rho$ sind äquivalent zu $m_B \in B_R(m_A|_\rho)$ bzw. $h_B \in B_R(h_A|_\rho)$.

Da B eindeutig für adt_F ist, enthält $B_R(m_A|_\rho)$ bzw. $B_R(h_A|_\rho)$ maximal ein Element und damit enthalten auch $B_R(m_A)$ und $B_R(h_A)$ maximal ein Element. ■

Das klassische Extension Lemma ist die eindeutige Übertragung von streng persistenten, freien Funktoren zwischen $Mod(adt_F)$ und $Mod(adt_B)$ eines parametrisierten abstrakten Datentyps auf streng persistente, freie Funktoren von $Mod(adt_A)$ nach $Mod(B(adt_A))$ für zulässige aktuelle Parameter adt_A. Für die Gleichungslogik findet man dieses Extension Lemma in [EM85], wo es einen Kernsatz für die Korrektheit von parametrisierten Spezifikationen darstellt und wichtig für die Komposition der Semantik parametrisierter Spezifikationen ist.

Satz 5.12 (Extension Lemma für Funktoren) *Existieren in der Institution $\mathcal{I}$ eindeutige amalgamierte Summen, dann gilt:*

Ist $B(X{:}\,adt_F)_\beta{:}\,adt_B$ ein parametrisierter abstrakter Datentyp und F ein Funktor von $Mod(adt_F)$ nach $Mod(adt_B)$, dann gilt:

1. *Ist F streng persistent bzgl. des Funktors $Mod(\beta)$, dann existiert für jeden zulässigen aktuellen Paramter adt_A ein eindeutiger, streng persistenter Funktor $F'{:}\,Mod(adt_A) \to Mod(B(adt_A))$ bzgl. $Mod(\beta')$, für den gilt:*

$$Mod(\rho') \circ F' = F \circ Mod(\rho).$$

2. *Ist F streng persistent und surjektiv, so auch F'.*

3. *Ist F streng persistent und links adjungierter Funktor zu $Mod(\beta)$, dann ist F' links adjungierter Funktor zu $Mod(\beta')$.*

Beweis

1. Definiere

$$\begin{aligned} F'(m) &:= m +_{m|_\rho} F(m|_\rho) \\ \text{und } F'(h) &:= h +_{h|_\rho} F(h|_\rho). \end{aligned}$$

Die amalgamierte Summe, und damit F', ist definiert, wenn F streng persistent bzgl. des Vergißfunktors $Mod(\beta)\colon Mod(adt_B) \to Mod(adt_F)$ ist, da nur dann

$$F(m|_\rho)|_\beta = m|_\rho$$

ist. Aufgrund der Rechenregeln für amalgamierte Summen (Lemma 5.9) ist F' auch ein Funktor.

Die strenge Persistenz von F' folgt aus der Definition der amalgamierten Summe, denn es gilt, daß

$$F'(m)|_{\beta'} = (m +_{m|_\rho} F(m|_\rho))|_{\beta'} = m$$

und

$$F'(h)|_{\beta'} = (h +_{h|_\rho} F(h|_\rho))|_{\beta'} = h$$

ist.

Die Eindeutigkeit von F' ergibt sich aus der Eindeutigkeit der amalgamierten Summe $F'(m) = m +_{m|_\rho} F(m|_\rho)$.

2. Sei nun F ein streng persistenter, surjektiver Funktor und sei m' aus $Mod(B(adt_A))$, dann existiert zu $m_B := m'|_{\rho'}$ ein m_F aus $Mod(adt_F)$ mit $F(m_F)$ ist gleich m_B. Da F streng persistent ist, ist $(m'|_{\beta'})|_\rho$ gleich $m_B|_\beta$ gleich m_F. Damit ist die amalgamierte Summe $m'|_{\beta'} +_{m_F} m_B$ definiert und somit ist $F'(m'|_{\beta'})$ gleich m'.

3. Sei nun F ein streng persistenter freier Funktor bzgl. des Vergißfunktors $Mod(\beta)$, dann ist zu zeigen daß F' ein freier Funktor bzgl. des Vergißfunktors $Mod(\beta')\colon Mod(B(adt_A)) \to Mod(adt_A)$ ist. Wegen der strengen Persistenz von F' ist die Einheit der Adjunktion die Identität, d.h. $\eta_m = id_m$ für alle m aus $Mod(adt_A)$.

Sei m eine Struktur in $Mod(adt_A)$ und m' eine Struktur in $Mod(B(adt_A))$ und $f\colon m \to m'|_{\beta'}$ ein Morphismus in $Mod(adt_A)$, dann ist zu zeigen, daß ein eindeutiger Morphismus $\tilde{f}$ von $F'(m)$ nach m' existiert für den gilt $f = \tilde{f}|_{\beta'}\eta_m = \tilde{f}|_{\beta'}$. Der Morphismus $f|_\rho$ ist ein Morphismus von $m|_\rho$ nach $(m'|_\beta)|_\rho$. Da F ein freier Funktor bzgl. $Mod(\beta)$ ist, existiert ein eindeutiger Morphismus $\tilde{f|_\rho}\colon F(m|_\rho) \to m'|_\beta$. Da die Einheit der Adjunktion für F ebenfalls die Identität ist, gilt, daß $\tilde{f|_\rho}|_\beta$ gleich $f|_\rho$ ist und damit ist die amalgamierte Summe definiert. Wegen der Eindeutigkeit der amalgamierte Summe ist dann

$$\tilde{f} := f +_{f|_\rho} \tilde{f|_\rho}$$

definiert und eindeutig.

■

Bei der Konstruktion von F' mithilfe der amalgamierten Summe muß F streng persistent sein, da sonst die amalgamierte Summe nicht definiert ist.

Kapitel 6

Verträglichkeit mit initialer Semantik

Bei der Verträglichkeit der initialen Semantik mit parametrisierten abstrakten Datentypen ist es die Aufgabe für eine parametrisierte Spezifikation $B(X\colon spec_F)_\beta\colon spec_B$ einen parametrisierten abstrakten Datentyp $B(X\colon adt_F)_\beta\colon adt_B$ zu finden, so daß für alle zulässigen aktuellen Parameter $spec_A$ von B

$$init(B(spec_A)) = B(init(spec_A))$$

gilt.

Aus der Kategorientheorie sind die universellen Morphismen $\langle c, \eta\colon c \to V_F(d)\rangle$ zu einem Funktor $V_F\colon \mathbf{D} \to \mathbf{C}$ ($c \in \mathbf{C}$ und $d \in \mathbf{D}$) bekannt, die die Eigenschaft haben zu jedem Morphismus $f\colon c \to V_F(d')$ einen eindeutigen Morphismus $\tilde{f}\colon d \to d'$ zu konstruieren (s.a. Definition 1.9). Das bedeutet, daß wenn c ein initiales Objekt der Kategorie $\mathbf{C}$ ist, d auch ein initiales Objekt in $\mathbf{D}$ sein muß.

Enthält $B(adt_A)$ nur Modelle m, für die es einen universellen Morphismus $\langle m_A, \eta\colon m_A \to m|_{\beta'}\rangle$ gibt, wobei $Mod(\beta')\colon Mod(B(spec_A)) \to Mod(spec_A)$, $m_A \in Mod(spec_A)$ und $m \in Mod(B(spec_A))$ ist, dann folgt aus der Initialität von m_A in $Mod(spec_A)$, daß m ein initiales Objekt in $Mod(B(spec_A))$ ist.

Definition 6.1 (Streng β–frei) *Ist $\beta\colon spec \to spec_1$ ein Spezifikationsmorphismus und m_1 eine Struktur aus $Mod(spec_1)$, dann ist m_1* streng β–frei*, gdw. es einen universellen Morphismus $\langle m_1|_\beta, id_{m_1|_\beta}\colon m_1|_\beta \to m_1|_\beta\rangle$ gibt.*

Ist m_1 streng β–frei, dann ist m_1 eine streng persistente, freie Erweiterung von $m_1|_\beta$ [EGL89]. Die strenge β–Freiheit ist angelehnt an die Definition der β-Freiheit [GB90] bzw. der $Mod(\beta)$–Freiheit [WE85]. Für die β–Freiheit bzw. $Mod(\beta)$–Freiheit einer Struktur m_1 werden alle Adjunktionen $\langle F, Mod(\beta), \eta\rangle\colon Mod(spec) \to Mod(spec_1)$ betrachtet und es wird verlangt, daß m_1 gleich $F(m)$ für ein m aus $Mod(spec)$ ist und daß $F(m)|_\beta$ isomorph zu m ist. Die in dieser Arbeit definierte strenge β–Freiheit für m_1 fordert nicht die Existenz von Adjunktionen, sondern nur die Existenz eines universellen Morphismus $\langle m_1|_\beta, \eta\colon m_1|_\beta \to m_1|_\beta\rangle$. Zusätzlich wird allerdings gefordert, daß η die Identität ist und nicht bloß ein Isomorphismus.[1]

Die Identität von η wird für das folgende Extension Lemma benötigt. Als Folge des Extension Lemmas, braucht man nicht mehr alle Modelle aus $B(adt_A)$ auf strenge β'–Freiheit zu testen, sondern es reicht aus das für die Modelle von adt_B zu tun.

[1] s.a. die Bemerkung zur Definition der Persistenz auf Seite 12

Lemma 6.2 (Extension Lemma streng β–frei) *Existieren in der Institution $\mathcal{I}$ eindeutige amalgamierte Summen, dann gilt:*

Ist $B(X\!:spec_F)_\beta\!:spec_B$ eine parametrisierte Spezifiaktion, $spec_A$ ein zulässiger aktueller Parameter für B und $m_B \in Mod(spec_B)$ streng β–frei, dann ist jedes Modell m von $B(spec_A)$, mit $m|_{\rho'}$ gleich m_B, streng β'–frei.

Beweis Gibt es zu einem $m' \in Mod(B(spec_A))$ einen Morphismus $f_A\!: m|_{\beta'} \to m'|_{\beta'}$, dann ist zu zeigen, daß ein eindeutiger Morphismus $\tilde{f}\!: m \to m'$ mit $\tilde{f}|_\beta = f_A$ existiert. Da $m_B := m|_{\rho'}$ streng β–frei ist und $f_F := f_A|_\rho$ ein Morphismus von $m_B|_\beta = (m|_{\beta'})|_\rho$ nach $(m'|_{\rho'})|_\beta$ ist, existiert ein eindeutiger Morphismus $\tilde{f_F}\!: m_B \to m'|_{\rho'}$ mit $\tilde{f_F}|_\beta = f_F$.

Mit der Definition

$$\tilde{f} := f_A +_{f_F} \tilde{f_F}$$

ist $\tilde{f}\!: m \to m'$ ein eindeutiger Morphismus zu f_A, da $\tilde{f_F}$ eindeutig zu f_F ist und die amalgamierte Summe eindeutig ist. ■

Das folgende Lemma ist die Kernaussage für die Verträglichkeit von parametrisierten abstrakten Datentypen mit der initialen Semantik. Aus der Eigenschaft, daß die strenge β–Freiheit ein Spezialfall eines universellen Morphismus ist und nach Satz 1.10, folgt das Lemma:

Lemma 6.3 *Ist $\beta\!: spec \to spec_1$ ein Spezifikationsmorphismus, $m_1 \in Mod(spec_1)$ eine streng β–freie Struktur und $m_1|_\beta$ ein initiales Objekt in der Kategorie $Mod(spec)$, dann ist m_1 ein initiales Objekt in der Kategorie $Mod(spec_1)$.*

Für den Beweis der Verträglichkeit wird noch benötigt, daß strenge β–Freiheit abgeschloßen ist gegen die Isomorphie von Strukturen.

Lemma 6.4 *Ist $\beta\!: spec \to spec'$ ein Spezifikationsmorphismus, $m' \in Mod(spec')$ streng β–frei und $m'_1 \in Mod(spec')$ isomorph zu m', dann ist m'_1 ebenfalls streng β–frei.*

Beweis Es ist zu zeigen, daß m'_1 streng β–frei ist, d.h. zu einem Morphismus $f\!: m'_1|_\beta \to m'_2|_\beta$ aus $Mod(spec)$ gibt es einen eindeutigen Morphismus $\tilde{f}\!: m'_1 \to m'_2$ aus $Mod(spec')$ mit $\tilde{f}|_\beta = f$.

Da m' streng β–frei und via ι isomorph zu m'_1 ist, existiert zu

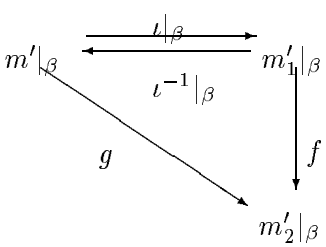
$$g := f\iota|_\beta$$

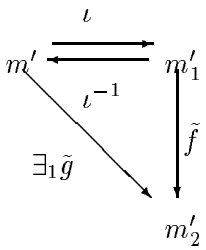
ein Morphismus $\tilde{g}\!: m' \to m'_2$ mit $\tilde{g}|_\beta = g$. Definiere

$$\tilde{f} := \tilde{g}\iota^{-1},$$

dann ist das Redukt von $\tilde{f}$ via β gleich

$$\tilde{f}|_\beta = (\tilde{g}\iota^{-1})|_\beta = f.$$





Um die Eindeutigkeit von f zu zeigen, nehmen wir an, es gibt einen zweiten Morphismus $h\colon m'_1 \to m'_2$ mit $h|_\beta = f$. Dann definiert

$$h' := h\iota$$

einen Morphismus von m' nach m'_2 mit

$$h'|_\beta = (h\iota)|_\beta = g.$$

Da $\tilde{g}$ eindeutig mit der Eigenschaft $\tilde{g}|_\beta = g$ ist, muß $h' = h\iota$ gleich $\tilde{g}$ sein und damit $h = \tilde{f}$. ■

Satz 6.5 (Verträglichkeit initialer Semantik mit Parametrisierung) *Existieren in der Institution $\mathcal{I}$ eindeutige amalgamierte Summen, dann gilt:*

Ist $B(X\colon spec_F)_\beta\colon spec_B$ eine für $spec_F$ konsistente parametrisierte Spezifikation und $spec_A$ ein zulässiger aktueller Parameter, dann gilt

$$B(init(spec_A)) = init(B(spec_A))$$

für einen parametrisierten abstrakten Datentyp $B(X\colon adt_F)_\beta\colon adt_B$ mit:

1. $adt_F = loose(spec_F)$,
2. $adt_B = \langle Sign(spec_B), \{m_B \mid m_B \in Mod(spec_B), m_B \text{ ist streng } \beta\text{–frei}\}\rangle$,
3. *B ist konsistent für adt_F,*
4. *und $Mod(init(spec_A))$ ist nicht leer.*

Beweis

($\subseteq$) Es ist zu zeigen, daß, wenn $m \in Mod(B(init(spec_A)))$ ist, dann ist m initiales Objekt in $Mod(B(spec_A))$. Wegen dem Amalgamierungs Lemma läßt sich m als amalgamierte Summe

$$m = m_A +_{m_F} m_B$$

darstellen. Da m_B in $Mod(adt_B)$ ist, ist m_B streng β–frei und nach dem Extension Lemma 6.2 ist dann m streng β'–frei. Dann ist aber nach Lemma 6.3 m initales Objekt in $Mod(B(spec_A))$, da $m|_{\beta'} = m_A$ initiales Objekt in $Mod(spec_A)$ ist.

($\supseteq$) Es ist zu zeigen, daß wenn m initiales Objekt in $Mod(B(spec_A))$ ist, dann ist m in $Mod(B(init(spec_A)))$.

Ist $Mod(init(spec_A))$ nicht leer, dann gibt es, da B konsistent ist, eine Struktur m' aus $Mod(B(init(spec_A)))$, die nach Teil 1 initiales Objekt in $Mod(B(spec_A))$ und damit isomorph zu m ist. Da Funktoren Isomorphie erhalten, sind $m'|_{\beta'}$ und $m|_{\beta'}$ isomorph zueinander und da $m'|_{\beta'}$ ein initiales Objekt in $Mod(spec_A)$ ist, ist $m|_{\beta'}$ auch ein initiales Objekt in $Mod(spec_A)$.

Außerdem ist $m'|_{\rho'}$ streng β–frei und da $m'|_{\rho'}$ isomorph zu $m|_{\rho'}$ ist, ist nach Lemma 6.4 auch $m|_{\rho'}$ streng β–frei. Damit ist m in $Mod(B(init(spec_A)))$. ■

Die Voraussetzung, daß $adt_F = loose(spec_F)$ ist garantiert, daß jede initiale Semantik einer aktuellen Parameterspezifikation auch ein aktueller Parameter des parametrisierten abstrakten Datentyps ist. Die Modelle von adt_F kann man auch geeignet einschränken, wenn man nicht alle initialen Semantiken von aktuellen Parameterspezifikationen als aktuelle Parameter des parametrisierten abstrakten Datentyps zulassen will. Für initiale Semantiken die zulässig für den parametrisierten abstrakten Datentyp sind gilt immer noch

$$B(init(spec_A)) = init(B(spec_A)).$$

Beispiel Eine Anwendungsmöglichkeit z.B. in der Gleichungslogik ist die Verwendung des Sorte **bool** mit ihrer üblichen Bedeutung als Menge der Wahrheitswerte true und false, in einer formalen Parameterspezifikation. Alleine mit Gleichungen läßt sich nicht verhindern, daß eine aktuelle Parameterspezifikation zulässig ist, die die Gleichung $true = false$ enthält und damit die Standardbedeutung von **bool** zerstört. Das diese Spezifikation nicht mehr zulässig ist kann man dadurch erreichen, daß die Modelle in adt_F nur noch diejenigen aus $loose(spec_F)$ sind, bei denen **bool** der Standardinterpretation entspricht. (s.a. [WE85, Ehr89, OSC89])

In der Gleichungslogik kann man zu jeder Algebra A_F aus $Mod(spec_F)$ eine Algebra A_B aus $Mod(spec_B)$ konstruieren, so daß $\langle A_A, \eta\colon A_A \to A_B|_\beta \rangle$ ein universeller Morphismus für geeignetes η ist [EM85, EGL89]. Damit η die Identität wird und damit A_B streng β–frei ist, müssen die Bedingungen „no–junk“ und „no–confusion“ erfüllt sein.

1. „no–junk“ bedeutet, daß bei der Konstruktion von A_B keine neuen Elemente aus Parametersorten erzeugt werden. Eine Bedingung dafür ist die hinreichende Vollständigkeit [Gan83, Pad85]. Die hinreichende Vollständigkeit von β verlangt, daß es für jeden Term $t \in T_{\Sigma_B}(V_{sorts(\Sigma_F)})_{sorts(\Sigma_F)}$ einen Term t' aus $T_{\Sigma_F}(V)$ gibt, so daß aus den Gleichungen von $spec_B$ die Gleichheit $t = \beta(t')$ folgt.

2. „no-confusion“ bedeutet, daß die Gleichungen in der Rumpfspezifikation keine Elemente aus Parametersorten miteinander identifizieren. Das heißt, sind t und t' zwei Terme aus $T_{\Sigma_F}(V)$ und folgt aus den Gleichungen von $spec_B$, daß $t = t'$ ist, dann muß $t = t'$ auch schon aus den Gleichungen des formalen Parameters folgen [Gan83, Pad85].

Aus der Existenz einer streng β–freien Algebra A_B zu jeder Algebra des formalen Parameters folgt die Konsistenz von B für adt_F.

Die letzte Bedingung für die Verträglichkeit ist in der Gleichungslogik immer erfüllt, da, wie aus dem Abschnitt 3.3 über die Semantik von Spezifikationen hervorgeht, zu jeder Spezifikation $spec_A$ ein initiales Modell existiert.

Die Konsequenz aus den letzten Ausführungen ist, daß immer wenn nach [EM85] die Parameterübergabe korrekt für eine parametrisierte Spezifikation und eine Argumentspezifikation ist, dann gelten die Voraussetzungen für die Verträglichkeit der initialen Semantik mit der Parametrisierung.

Teil III

Stabilität und Implementierung

Kapitel 7

Abstraktorimplementierung

Abstrakte Datentypen bestehen aus einer Signatur und einer Menge von Modellen. Im Prozeß der Softwareentwicklung wird die Menge der Modelle des abstrakten Datentyps durch Designentscheidungen immer weiter reduziert, bis im Idealfall nur noch eine Realisierung der Signatur, also ein Programm, übrig bleibt. Dieser Prozeß, auch Refinement oder Verfeinerung genannt [ST86, Wir89], ist die Basis für die Abstraktorimplementierung.

Definition 7.1 (Verfeinerung (Refinement)) *Ein abstrakter Datentyp adt wird zu einem abstrakten Datentyp* adt_1 verfeinert

$$adt \rightsquigarrow adt_1,$$

wenn adt und adt_1 *die gleiche Signatur haben und die Kategorie* $Mod(adt_1)$ *eine Unterkategorie von* $Mod(adt)$ *ist*

$$adt_1 \subseteq adt.$$

Die Bedingung ist äquivalent zu der Forderung, daß es einen **Adt**–Morphismus ι von adt nach adt' gibt, der auf den Signaturen die Identität ist.

Neben dem Einschränken der Modelle eines abstrakten Datentyps muß es auch möglich sein die Modelle des abstrakten Datentyps mithilfe der Modelle eines anderen abstrakten Datentyps, der z.B. eine reichere Signatur hat, zu konstruieren. Ein anderer Aspekt ist, daß die Implementierung nicht exakt den abstrakten Datentyp implementiert sondern nur sein Verhalten, also eine Abstraktion des abstrakten Datentyps. Diese zwei Anforderungen sind im folgenden Implementierungsbegriff zusammengefaßt.

Definition 7.2 (Abstraktorimplementierung 1) *Ein abstrakter Datentyp* adt_1 *implementiert den abstrakten Datentyp adt bzgl. des Abstraktors* α *via* $\sigma: Sign(adt) \to Sign(adt_1)$

$$adt \overset{\alpha}{\underset{\sigma}{\rightsquigarrow}} adt_1,$$

gdw. $adt_1|_\sigma$ *eine Verfeinerung der Abstraktion von adt bzgl.* α *ist*

$$Beh_\alpha(adt) \rightsquigarrow adt_1|_\sigma.$$

Mit anderen Worten σ *muß ein* **Adt***–Morphismus von* $Beh_\alpha(adt)$ *nach* adt_1 *sein.*

Die Verhaltensimplementierung parametrisierter abstrakter Datentypen wird punktweise auf die Verhaltensimplementierung abstrakter Datentypen zurückgeführt.

Definition 7.3 (Abstraktorimplementierung 2)

Ein parametrisierter abstrakter Datentyp $B_1(X\!:\!adt_{F_1})_{\beta_1}\!:\!adt_{B_1}$ *implementiert den parametrisierten abstrakten Datentyp* $B(X\!:\!adt_F)_\beta\!:\!adt_B$ *bzgl. des Abstraktors* α *via* $\sigma\!:\!Sign(adt_B) \to Sign(B_1(adt_F))$,

$$B \overset{\alpha}{\underset{\sigma}{\rightsquigarrow}} B_1$$

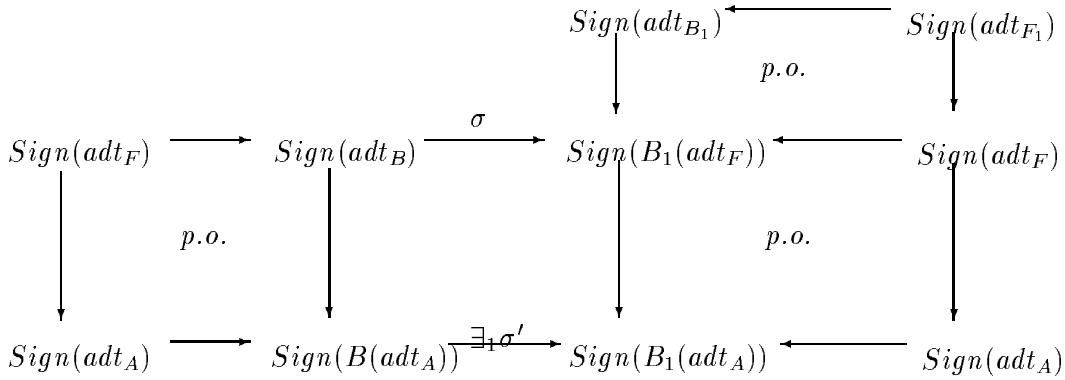
gdw. für alle zulässigen aktuellen Parameter adt_A *von* B *die Anwendung von* B_1 *auf* adt_A *eine Implementierung der Anwendung von* B *auf* adt_A *bzgl. des Abstraktors* α' *via* σ' *ist*

$$B(adt_A) \overset{\alpha'}{\underset{\sigma'}{\rightsquigarrow}} B_1(adt_A).$$

Der Abstraktor α' *entsteht aus* α *durch Übersetzung mit* ρ'

$$\alpha' := \rho'(\alpha).$$

Für jeden zulässigen Parameter adt_A *ist* σ' *die eindeutige Erweiterung von* σ *auf einen Signaturmorphismus von* $Sign(B(adt_A))$ *nach* $Sign(B_1(adt_A))$.



Im Gegensatz zu anderen Implementierungskonzepten [ST86, Wir89, Hen89b] wird nicht gefordert, daß die formalen Parameter von B und B_1 gleich sein oder die gleiche Signatur haben müssen. Es muß nur jeder zulässige Parameter für B auch ein zulässiger Parameter für die Implementierung B_1 sein. Damit ist es möglich die Anforderung an die aktuellen Parameter der Implementierung abzuschwächen. Entsprechend ist σ auch kein Morphismus von $Sign(adt_B)$ nach $Sign(adt_{B_1})$ sondern ein Morphismus von $Sign(adt_B)$ nach $Sign(B_1(adt_F))$.

Der in dieser Arbeit verwendete Begriff der Abstraktorimplementierung ist ein Spezialfall des Begriffs aus [ST86], da hier nur die Ableitung eines abstrakten Datentyps als Konstruktor zugelassen wird, während in [ST86] alle Operationen auf abstrakten Datentypen zugelassen sind, die mithilfe von Funktionen auf Strukturen definiert sind.[1]

Die Einschränkung auf die Ableitung als Konstruktor geschieht zur Konzentration auf die Abstraktoren und deren Beziehung zur Parametrisierung.

[1] die Ableitung von abstrakten Datentypen ist z.B. ein Konstruktor im Sinne von [ST86], die Übersetzung von abstrakten Datentypen dagegen nicht.

7.1 Vertikale Komposition

Damit ein Implementierungsbegriff praktisch anwendbar ist, verlangt man von ihm folgende Eigenschaften [ST86, Wir89]:

- Die Implementierung einer Implementierung eines abstrakten Datentyps muß ebenfalls eine Implementierung des abstrakten Datentyps sein. Diese Eigenschaft ist die vertikale Komposition von Implementierungen.
- Ein abstrakter Datentyp muß an jeder Stelle durch seine Implementierung ersetzt werden können.[2] Diese Eigenschaft wird horizontale Komposition genannt.

Dieser und der nächste Abschnitt befaßt sich mit den Bedingungen für die vertikale und horizontale Komposition.

Satz 7.4 (vertikale Komposition 1) *Ist adt_1 eine Implementierung der Abstraktion von adt und adt_2 eine Implementierung der Abstraktion von adt_1*

$$adt \overset{\alpha}{\underset{\sigma}{\rightsquigarrow}} adt_1 \overset{\alpha_1}{\underset{\sigma_1}{\rightsquigarrow}} adt_2,$$

dann ist adt_2 eine Implementierung der Abstraktion von adt

$$adt \overset{\alpha}{\underset{\sigma\sigma_1}{\rightsquigarrow}} adt_2,$$

wenn $\equiv_{\alpha_1} \subseteq \sigma(\equiv_\alpha)$ ist.

Beweis Da adt_2 die Abstraktion von adt_1 implementiert, ist $Mod(adt_2)|_{\sigma_1}$ eine Unterkategorie von $Mod(Beh_{obs_1}(adt_1))$

$$adt_2|_{\sigma_1} \subseteq Beh_{\alpha_1}(adt_1).$$

Da $Mod(\sigma)$ monoton ist erhält man, daß $Mod(adt_2)|_{\sigma\sigma_1}$ eine Unterkategorie des σ-Redukts der Modelle von $Beh_{\alpha_1}(adt_1)$ ist

$$adt_2|_{\sigma\sigma_1} \subseteq Beh_{\alpha_1}(adt_1)|_\sigma.$$

Da adt_1 die Abstraktion von adt implementiert, ist

$$adt_1|_\sigma \subseteq Beh_\alpha(adt)$$

und da α monoton und idempotent ist, erhält man

$$Beh_\alpha(adt_1|_\sigma) \subseteq Beh_\alpha(adt).$$

Somit bleibt zu zeigen, daß

$$Beh_{\alpha_1}(adt_1)|_\sigma \subseteq Beh_\alpha(adt_1|_\sigma)$$

ist. Ist m aus $Mod(adt_1)$ und $m' \equiv_{\alpha_1} m$, dann gilt, da $\equiv_{\alpha_1}$ eine Teilmenge von $\sigma(\equiv_\alpha)$ ist, daß $m|_\sigma \equiv_\alpha m'|_\sigma$ gilt. Wegen der Definition von $adt_1|_\sigma$ ist $m|_\sigma$ in $Mod(adt_1|_\sigma)$, und damit liegt m' in $Mod(Beh_\alpha(adt_1|_\sigma))$. ■

[2] genauer: ist $adt \overset{\alpha}{\underset{\sigma}{\rightsquigarrow}} adt'$, dann muß $adt'|_\sigma$ an die Stelle von adt treten können

Satz 7.5 (vertikale Komposition 2) *Ist der parametrisierte abstrakte Datentyp B_1 eine Implementierung der Abstraktion des parametrisierten abstrakten Datentyps B und B_2 eine Implementierung der Abstraktion von B_1*

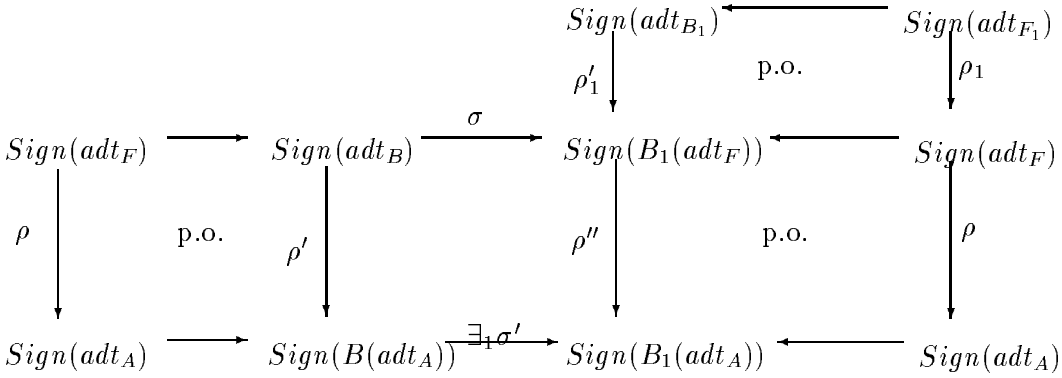
$$B \overset{\alpha}{\underset{\sigma}{\rightsquigarrow}} B_1 \overset{\alpha_1}{\underset{\sigma_1}{\rightsquigarrow}} B_2,$$

dann ist B_2 eine Implementierung der Abstraktion von B

$$B \overset{\alpha}{\underset{\sigma\sigma_1}{\rightsquigarrow}} B_2,$$

wenn $\rho_1'(\equiv_{\alpha_1}) \subseteq \sigma(\equiv_\alpha)$ ist, wobei der Morphismus ρ_1' der eindeutige Morphismus von adt_{B_1} nach $B_1(adt_F)$ zu dem Morphismus ρ_1 von adt_{F_1} nach adt_F ist.

Beweis Es ist zu zeigen, daß für jeden aktuellen Parameter adt_A von B die Anwendung von B_2 auf adt_A eine Implementierung der $\rho'(\alpha)$–Abstraktion von $B(adt_A)$ ist.



Da B_2 das Verhalten von B_1 implementiert, ist auch $B_2(adt_A)$ eine Implementierung der Abstraktion von $B_1(adt_A)$. Da B_1 das Verhalten von B implementiert, ist auch $B_1(adt_A)$ eine Implementierung der Abstraktion von $B(adt_A)$

$$B(adt_A) \overset{\rho'(\alpha)}{\underset{\sigma'}{\rightsquigarrow}} B_1(adt_A) \overset{\rho''\rho_1'(\alpha_1)}{\underset{\sigma_1'}{\rightsquigarrow}} B_2(adt_A),$$

Um Satz 7.4 anzuwenden, ist zu zeigen, daß

$$\rho''(\rho_1'(\equiv_{\alpha_1})) \subseteq \sigma'(\rho'(\equiv_\alpha))$$

gilt. Da $\rho_1'(\equiv_{\alpha_1})$ laut Voraussetzung eine Teilmenge von $\sigma(\equiv_\alpha)$ ist, ist $\rho''(\rho_1'(\equiv_{\alpha_1}))$ eine Teilmenge von $\rho''(\sigma(\equiv_\alpha)) = \sigma'(\rho'(\equiv_\alpha))$. Nach Satz 7.4 ist nun $B_2(adt_A)$ eine Implementierung der $\sigma'(\alpha)$–Abstraktion von $B(adt_A)$.

Wenn $\rho_1'(\equiv_{\alpha_1})$ eine Teilmenge von $\sigma(\equiv_\alpha)$ ist, dann ist auch $\rho''(\rho_1'(\equiv_{\alpha_1}))$ eine Teilmenge von $\rho''(\sigma(\equiv_\alpha)) = \rho''(\rho_1'(\equiv_\alpha))$ und dann ist wegen Satz 7.4 $B_2(adt_A)$ eine Implementierung der $\rho'(\alpha)$–Abstraktion von $B(adt_A)$. ■

7.2 Horizontale Komposition

Die Bedingung der horizontalen Komposition verlangt, daß an jeder Stelle, an der ein abstrakter Datentyp stehen kann auch seine Implementierung stehen kann.

Ist z.B. $B(X\!:\!adt_F)_\beta\!:\!adt_B$ ein parametrisierter abstrakter Datentyp, adt_A ein zulässiger Parameter für B und adt'_A eine Implementierung von adt_A bzgl. α_A via σ, dann muß auch $adt'_A|_\sigma$ ein zulässiger aktueller Parameter von B sein und $B(adt'_A|_\sigma)$ eine Verfeinerung der Abstraktion von $B(adt_A)$ bzgl. α' sein

$$B(adt_A) \overset{\alpha'}{\underset{\iota}{\rightsquigarrow}} B(adt'_A|_\sigma),$$

wobei $\alpha' = \rho'(\alpha_A)$ und α_B eine Abstraktor der Signatur $Sign(adt_B)$ ist.

Man kann nicht verlangen, daß $B(adt'_A|_\sigma)$ eine Verfeinerung von $B(adt_A)$ ist, da von dem abstrakten Datentyp $adt'_A|_\sigma$ nur verlangt wird, daß er eine Verfeinerung der Abstraktion von adt_A ist und nicht eine Verfeinerung von adt_A selbst. Für den Spezialfall der α_A–Abstraktion von adt_A als Implementierung der Abstraktion von adt_A

$$adt_A \overset{\alpha_a}{\underset{\iota}{\rightsquigarrow}} adt_A,$$

erhält man aus der Forderung der horizontalen Komposition, daß $B(Beh_{\alpha_A}(adt_A))$ eine Implementierung der α'–Abstraktion von $B(adt_A)$ sein muß, oder mit anderen Worten

$$(*) B(Beh_{\alpha_A}(adt_A)) \subseteq Beh_{\alpha'}(B(adt_A)).$$

Die Eigenschaft $(*)$ heißt auch Erhaltung der α_A–Äquivalenz von B („B preserves α_A–equivalence“ [ST86],) weil B zu $\equiv_{\alpha_A}$–äquivalenten Modellen aus $Mod(adt_A)$ nur $\equiv_{\alpha'}$–äquivalente Modelle konstruiert. Die Wichtigkeit von $(*)$ für die horizontale Komposition von Implementierungen zeigt sich in folgendem Satz, der besagt, daß $(*)$ hinreichende Bedingung für die horizontale Komposition ist.

Satz 7.6 (horizontale Komposition) *Gegeben ist ein parametrisierter abstrakter Datentyp $B(X\!:\!adt_F)_\beta\!:\!adt_B$, ein zulässiger aktueller Parameter adt_A für B und ein Abstraktor α_B der Signatur $Sign(adt_B)$.*

Für jede Implementierung adt'_A der α_A–Abstraktion von adt_A

$$adt_A \overset{\alpha_A}{\underset{\sigma}{\rightsquigarrow}} adt'_A$$

ist $B(adt'_A|_\sigma)$ eine Implementierung der α'–Abstraktion von $B(adt_A)$, wobei $\alpha' := \rho'(\alpha_B)$ ist, falls

$$B(Beh_{\alpha_A}(adt_A)) \subseteq Beh_{\alpha'}(B(adt_A))$$

ist.

Beweis Es ist zu zeigen, daß

$$B(adt'_A|_\sigma) \subseteq Beh_{\alpha'}(B(adt_A))$$

ist. Da adt'_A eine Implementierung von adt_A und B monoton ist, erhält man

$$B(adt'_A|_\sigma) \subseteq B(Beh_{\alpha_A}(adt_A)).$$

Aus der Voraussetzung, daß $B(Beh_{\alpha_A}(adt_A)) \subseteq Beh_{\alpha'}(B(adt_A))$ ist, folgt dann

$$B(adt'_A|_\sigma) \subseteq Beh_{\alpha'}(B(adt_A)).$$

■

Das Erhalten der α_A–Äquivalenz ist nur dann gewährleistet, wenn die Abstraktion des aktuellen Parameters adt_A bzgl. α_A ebenfalls ein zulässiger Parameter ist. Dies führt zu dem Begriff der Zulässigkeit bzgl. eines Abstraktors (s.a. Kapitel 9).

Kapitel 8

Beispiele für Abstraktoren

8.1 Verhaltensäquivalenz

Die Definiton der Verhaltensäquivalenz war gedacht als Erweiterung des Konzepts der beobachtbaren Sorten (Abschnitt 8.2) auf beliebige Institutionen [ST85b, ST85a].[1] Die Vorstellung hinter der Tatsache, daß zwei Strukturen verhaltensäquivalent sind ist, daß sie auf Beobachtung gleich reagieren. Beobachtungen sind Formeln und die Ergebnisse von Beobachtungen sind die Festellungen, ob die Formeln in einer Struktur gültig sind oder nicht. Demnach zeigen zwei Strukturen dasselbe Verhalten, wenn für jede Formel einer Menge von Beobachtungen, die Formel in beiden Strukturen gültig oder nicht gültig ist.

Um z.B. in der Gleichungslogik Beobachtungen über Elemente anzustellen, die sich nicht durch Grundterme repräsentieren lassen, können Beobachtungen auch freie Variablen enthalten (s.a. Abschnitt 2.3)

Definition 8.1 (Beobachtung) *Eine Σ–Beobachtung, oder einfach Beobachtung, ist ein offene Σ–Formel $\langle \phi, e \rangle$.*

Definition 8.2 (Verhaltensäquivalenz) *Sind m und m_1 zwei Strukturen aus $Mod(\Sigma)$ und obs eine Menge von Σ–Beobachtungen, dann ist m reduzierbar auf m_1 bzgl. der Beobachtungen in obs*

$$m \leq_{obs} m_1,$$

gdw. wenn es für jede Beobachtung $o = \langle \phi, e \rangle$ aus obs und jede Variablenbelegung m' mit Werten aus m eine Variablenbelegung m'_1 mit Werten aus m_1 gibt, so daß

$$m \models_{m'} o \Leftrightarrow m_1 \models_{m'_1} o$$

gilt.

Die Strukturen m und m_1 zeigen das gleiche Verhalten (sind verhaltensäquivalent) bzgl. der Beobachtungen in obs

$$m \equiv_{obs} m_1$$

gdw.

$$m \leq_{obs} m_1 \text{ und } m_1 \leq_{obs} m.$$

[1] Dazu, daß dies nicht in vollem Umfang gelungen ist, s.a. Abschnitt 8.2. Der Grund dafür ist, daß sich Vehaltensäquivalenz und Äquivalenz bzgl. beobachtbarer Sorten bei der Übersetzung mit Signaturmorphismen unterschiedlich verhalten.

Die Definition der Reduzierbarkeit $m \leq_{obs} m_1$ stammt im wesentlichen aus [ST84, ST85b]. Der Unterschied zu der Definition dieser Arbeit besteht darin, daß in [ST84] nur offene Formeln über einer festen Variablenmenge betrachtet werden und die Definition von $m \leq_{obs} m_1$ verlangt, daß für alle Variablenbelegungen m' eine Variablenbelegung m'_1 existiert, so daß für alle $o \in obs$ gilt:

$$m \models_{m'} o \Leftrightarrow m_1 \models_{m'_1} o.$$

Dabei geht die Unabhängigkeit der Beobachtungen voneinander verloren. Gilt z.B. $m \leq_{obs} m_1$ und $m \leq_{\{o\}} m_1$, dann braucht nicht $m \leq_{obs \cup \{o\}} m_1$ zu gelten, wenn o über dieselben freien Variablen wie obs aufgebaut ist. Die Definition der Reduzierbarkeit $m \leq_{obs} m_1$ in dieser Arbeit dagegen betrachtet jede Formel aus obs mit ihren freien Variablen unabhängig voneinander, als ob ihre Variablenmengen alle disjunkt wären. Im Grenzfall, wenn die Formeln keine freien Variablen enthalten, sind beide Definitionen äquivalent.

Korollar 8.3 *Ist für alle Σ–Formeln $\langle \phi, e \rangle$ aus obs der Signaturmorphisus ϕ die Identität auf Σ, dann ist e in $Sen(\Sigma)$ und $\langle \phi, e \rangle$ wird mit e identifiziert. Die Definition der Verhaltensäquivalenz bzgl. obs zweier Strukturen m und m_1 aus $Mod(\Sigma)$ vereinfacht sich dann zu:*

$$m \equiv_{obs} m_1$$

gdw.

$$\forall \langle \phi, e \rangle \in obs : m \models e \Leftrightarrow m_1 \models e.$$

Beweis Ist $\langle \phi, e \rangle$ eine offene Σ–Formel und $\phi = id_\Sigma$, die Identität auf Σ, dann ist für m aus $Mod(\Sigma)$ die Struktur m die einzige Struktur für die $m|_{id_\Sigma} = m$ gilt. Das heißt für alle Variablenbelegungen m' mit Werten aus m gilt, daß $m \models_{m'} \langle \phi, e \rangle$ gdw. $m \models e$.

Damit erhält man, daß $m \leq_{obs} m_1$ genau dann gilt, wenn für alle $\langle \phi, e \rangle$ aus obs aus der Gültigkeit von e in m die Gültigkeit von e in m_1 folgt.

Es gilt nun $m \equiv_{obs} m_1$ gdw. $m \leq_{obs} m_1$ und $m_1 \leq_{obs} m$ gilt, d.h. wenn man für alle $\langle \phi, e \rangle$ aus obs

$$m \models e \Leftrightarrow m_1 \models e$$

hat. ■

Aus der Definiton von $\leq_{obs}$ folgen einige einfache Aussagen.

Fakt 8.4 *Sind obs_1 und obs_2 Σ–Beobachtungen, dann gilt:*

1. *Ist $obs_1 \subseteq obs_2$, dann ist $\leq_{obs_2} \subseteq \leq_{obs_1}$.*
2. $\leq_{obs_1} \cap \leq_{obs_2} = \leq_{obs_1 \cup obs_2}$.

Entsprechend erhält man für $\equiv_{obs}$ folgende Aussagen.

Fakt 8.5 *Sind obs_1 und obs_2 Mengen von Σ–Beobachtungen, dann gilt:*

1. *$\equiv_{obs_1}$ ist eine Äquivalenzrelation.*
2. *Ist $obs_1 \subseteq obs_2$, dann $\equiv_{obs_2} \subseteq \equiv_{obs_1}$.*
3. $\equiv_{obs_1} \cap \equiv_{obs_2} = \equiv_{obs_1 \cup obs_2}$.

Satz 8.6 *Unter der Voraussetzung, daß in der Institution $\mathcal{I}$ eindeutige amalgamierte Summen existieren gilt:*

Ist $\sigma: \Sigma \to \Sigma_1$ ein Signaturmorphismus und obs eine Menge von Σ–Beobachtungen dann ist

$$\leq_{\sigma(obs)} = \sigma(\leq_{obs}).$$

Beweis

($\Longrightarrow$) Für jede Beobachtung $\sigma(o)$ aus $\sigma(obs)$ ist zu zeigen, daß es für jede Variablenbelegung m' mit Werten aus m eine Variablenbelegung m_1' mit Werten aus m_1 gibt, so daß

$$m \models_{m'} o \Leftrightarrow m_1 \models_{m_1'} o$$

gilt.

Da $(m'|_{\sigma'})|_{\phi}$ gleich m ist, ist $m'|_{\sigma'}$ eine Variablenbelegung mit Werten aus m. Da

$$m|_{\sigma} \leq_{obs} m_1|_{\sigma}$$

gilt, existiert eine Variablenbelegung m_1'' mit Werten aus $m_1|_{\sigma}$ dessen Redukt auf Σ die Struktur $m_1|_{\sigma}$ ist und man erhält, daß

$$m|_{\sigma} \models_{m'|_{\sigma'}} o \Leftrightarrow m_1|_{\sigma} \models_{m_1''} o$$

gilt.

Da die Institution $\mathcal{I}$ eindeutig Strukturen konstruiert existiert die amalgamierte Summe

$$m_1' := m_1'' +_{m_1|_{\sigma}} m_1$$

und m_1' ist eine Variablenbelegung mit Werten aus m_1.

Man erhält nun wegen der Erfüllbarkeitsbedingung für offene Formeln daß

$$m \models_{m'} \sigma(o) \Leftrightarrow m_1 \models_{m_1'} \sigma(o)$$

gilt.

($\Longleftarrow$) Für jede Beobachtung o aus obs ist zu zeigen, daß es für jede Variablenbelegung m'' mit Werten aus $m|_{\sigma}$ eine Variablenbelegung m_1'' mit Werten aus $m_1|_{\sigma}$ gibt, so daß

$$m|_{\sigma} \models_{m''} o \Leftrightarrow m_1|_{\sigma} \models_{m_1''} o$$

gilt.

Ist m'' eine Variablenbelegung mit Werten aus $m|_{\sigma}$, dann ist $m''|_{\phi} = m|_{\sigma}$. Da die Institution $\mathcal{I}$ eindeutig Strukturen konstruiert, ist die amalgamierte Summe

$$m' := m'' +_{m''|_{\phi}} m$$

definiert und m' eine Variablenbelegung in m. Dann gibt es aber eine Variablenbelegung m_1' mit Werten aus m_1 so daß

$$m \models_{m'} \sigma(o) \Leftrightarrow m_1 \models_{m_1'} \sigma(o)$$

gilt. Definiere nun m_1'' als $m_1'|_{\sigma'}$, dann gilt

$$m|_{\sigma} \models_{m''} o \Leftrightarrow m_1|_{\sigma} \models_{m_1''} o,$$

da $m'|_{\sigma'}$ gleich m'' ist und die Erfüllbarkeitsbedingung für offene Formeln gilt. ■

Korollar 8.7 *Unter der Voraussetzung, daß in der Institution $\mathcal{I}$ eindeutige amalgamierte Summen existieren gilt:*

Ist $\sigma: \Sigma \to \Sigma_1$ ein Signaturmorphismus und obs eine Menge von Σ–Beobachtungen dann ist

$$\equiv_{\sigma(obs)} = \sigma(\equiv_{obs}).$$

Für den Fall, daß *obs* nur geschloßene Formeln enthält, d.h. Formeln $\langle \phi, e \rangle$, bei denen ϕ die Identität ist, braucht man die Eigenschaft, daß in der Institution $\mathcal{I}$ eindeutig amalgamierte Summen existieren, nicht für den Satz, da immer nur genau eine Variablenbelegung existiert.

Definition 8.8 ($cl(obs)$) *Zu Σ–Beobachtungen obs ist $cl(obs)$ definiert als die größte Menge von offenen Σ–Formeln für die gilt:*

$$\equiv_{cl(obs)} = \equiv_{obs}.$$

Die Definition von cl(obs) in dieser Arbeit unterscheidet sich von der aus [ST84]. Dort ist $cl(obs)$ der Abschluß von *obs* unter Negation, Konjunktion, logischer Äquivalenz und Allquantifizierung. Die Definition von $cl(obs)$ in dieser Arbeit ist dagegen nicht konstruktiv. Die Menge $cl(obs)$ enthält die Menge $cl(obs)$ aus [ST84]. Beide Mengen sind aber nicht gleich. Mit der Definition von $cl(obs)$ in dieser Arbeit folgt aus $\equiv_{obs} = \equiv_{obs'}$, daß $cl(obs) = cl(obs')$ gilt. Diese Eigenschaft gilt aber nicht in [ST84].

Satz 8.9 *Für eine Menge von Beobachtungen obs ist $cl(obs)$ die Vereinigung aller Mengen von Beobachungen obs', bei denen $\equiv_{obs} = \equiv_{obs'}$ ist*

$$cl(obs) = \bigcup_{\equiv_{obs} = \equiv_{obs'}} obs'.$$

Beweis

Einerseits ist zu zeigen, daß $\equiv_{cl(obs)} = \equiv_{obs}$ ist. Das gilt, da obs' so gewählt ist, daß $\equiv_{obs'} = \equiv_{obs}$ ist und damit

$$\bigcap \equiv_{obs'} = \equiv_{obs}$$

ist. Laut Fakt 8.5 erhält man dann

$$\begin{aligned} \bigcap \equiv_{obs'} &= \equiv_{\bigcup obs'} \\ &= \equiv_{cl(obs)}. \end{aligned}$$

Andererseits ist zu zeigen, daß $cl(obs)$ auch wirklich maximal ist. Annahme, es gibt ein obs' für das $\equiv_{obs'} = \equiv_{obs}$, gilt und obs' ist keine Teilmenge von $cl(obs)$. Das steht im Wiederspruch zu der Definition von $cl(obs)$, nach der die Beobachtungen obs' in $cl(obs)$ liegen müssen. ■

Fakt 8.10 (Eigenschaften von cl(obs)) *Sind obs und obs_1 Mengen von Σ–Beobachtungen, dann gilt:*

1. $obs \subseteq cl(obs)$
2. *Ist $\equiv_{obs}$ gleich $\equiv_{obs_1}$, dann ist auch $cl(obs)$ gleich $cl(obs_1)$*
3. $cl(cl(obs)) = cl(obs)$

Beweis

1. $obs \subseteq cl(obs)$ folgt aus der Tatsache, daß $\equiv_{obs} = \equiv_{cl(obs)}$ ist.
2. Da $\equiv_{obs} = \equiv_{obs_1}$ gilt, ist $cl(obs_1) \subseteq cl(obs)$ und $cl(obs) \subseteq cl(obs_1)$.
3. $cl(cl(obs)) = cl(obs)$ folgt aus dem vorherigen Teil mit $obs_1 = cl(obs)$.

■

Satz 8.11 *Ist obs eine Menge von Σ–Beobachtungen und M eine Menge von Strukturen aus $Mod(\Sigma)$, dann ist $\equiv_{obs}(M)$ eine Teilmenge der Modelle von $Th(M) \cap cl(obs)$*

$$\equiv_{obs}(M) \subseteq Mod(Th(M) \cap cl(obs)).$$

Beweis Ist m' aus $\equiv_{obs}(M)$, dann existiert ein m aus M mit $m' \equiv_{obs} m$. Es ist zu zeigen, daß die Schnittmenge der Theorie von M und $cl(obs)$ in m' gültig ist.

Ist e aus $Th(M) \cap cl(obs)$, dann ist e gültig in m und da $e \in cl(obs)$ ist, ist e wegen der Definition von $\equiv_{obs}$ auch in m' gültig. ■

Die Umkehrung des Satzes, also

$$Mod(E^* \cap cl(obs)) \subseteq \equiv_{obs}(Mod(E)),$$

gilt leider nicht. In [ST84] findet man den Beweis, daß in der Institution der Prädikatenlogik erster Stufe und für Beobachtungen ohne freie Variablen, also Beobachtungen der Form $\langle id_\Sigma, e\rangle$, gilt, daß

$$Mod(E^* \cap cl(obs)) = \equiv_{obs}(Mod(E))$$

ist. Das gilt aber schon dann nicht mehr, wenn man freie Variablen in Beobachtungen zuläßt [ST84].

Der Satz 8.11 ergibt eine korrekte, im allgemeinen aber nicht vollständige, Inferenzregel für die Theoreme von $\equiv_{obs}(M)$ [ST84, ST85b].

Ist e in $Th(M)$ und beobachtbar, d.h. $e \in cl(obs)$, dann ist e in allen Strukturen aus $\equiv_{obs}(M)$ gültig.

8.2 Äquivalenz bzgl. beobachtbarer Sorten

Einen weiteren Abstraktor erhält man, wenn einen nur die Ergebnisse von Berechnungen auf bestimmten Sorten interessieren. Alle Berechnungen mit Argumenten beobachtbarer Sorten und beobachtbarer Ergebnissorte sollen in äquivalenten Algebren bei gleicher Eingabe das gleiche Ergebnis liefern. Die Ergebnisse in unbebobachtbaren Sorten und mit Argumenten aus unbeobachtbarer Sorten interessieren nicht.

Im Stack–Beispiel aus der Einleitung interessiert einen nicht, ob $pop(push(s, e)) = s$ ist, sondern nur, daß die Elemente in der umgekehrten Reihenfolge vom Stack geholt werden können, in der sie auf den Stack abgelegt wurden. Das heißt es ist z.B. wichtig, daß

$$top(pop(push(pop(s, e_1), e))) = top(pop(s, e_1)) = e_1$$

ist. Demnach ist die Sorte stack unbeobachtbar, die Sorte der Elemente des Stacks dagegen ist beobachtbar.

Das Konzept der beobachtbaren Sorten ist allerdings nur in Institutionen sinnvoll, in denen es, wie z.B. in der Institution der Gleichungslogik, eine Vorstellung von Sorten gibt. Im

folgenden wird das Konzept der Äquivalenz bzgl. beobachtbarer Sorten in der Institution der Gleichungslogik entwickelt.

Eine Menge beobachtbarer Sorten *beob* ist eine Teilmenge der Sorten einer Signatur Σ der Gleichungslogik. Beobachtbare Terme sind Terme über Σ, deren Variablen von beobachtbarer Sorte sind und deren Ergebnissorte beobachtbar ist.

Definition 8.12 (beobachtbare Sorten) *Eine Menge beobachtbaren Sorten beob einer Signatur Σ der Gleichungslogik, ist ein Teilemenge der Sorten von Σ*

$$beob \subseteq sorts(\Sigma).$$

Definition 8.13 (beobachtbare Terme) *Eine beobachtbarer Term ist ein Term t der Signatur Σ der Gleichungslogik mit Variablen beobachtbarer Sorten, und als Ergebnissorte eine beobachtbare Sorte*

$$t \in T_\Sigma(V_{beob})_{beob}.$$

Zwei Algebren A und B sind äquivalent bzgl. beobachtbarer Sorten, wenn die Trägermengen beobachtbarer Sorten von A und B gleich sind und wenn alle beobachtbaren Terme mit gleichen Argumenten gleiche Ergebnisse in A und B liefern.

Definition 8.14 (beobachtbare Äquivalenz) *Zwei Σ–Algebren A und B sind beobachtbar äquivalent bzgl. der beobachtbaren Sorten $beob \subseteq sorts(\Sigma)$*

$$A \equiv_{beob} B,$$

wenn die Trägermengen der beobachtbaren Sorten A und B gleich sind

$$A_s = B_s \text{ für } s \text{ aus } beob$$

und für jeden beobachtbaren Term t aus $T_\Sigma(V_{beob})_{beob}$ und jede Variablenbelegung $b_A: V \to A$ und $b_B: V \to B$, die auf den Variablen aus V_{beob} gleich sind

$$b_A(x) = b_B(x) \text{ für } x \text{ aus } V_{s \in beob}$$

die Auswertung von t in A mit der Variablenbelegung b_A gleich der Auswertung von t in B mit der Variablenbelegung b_B ist

$$b_A^*(t) = b_B^*(t).$$

Die Definition der Äquivalenz bzgl. beobachtbarer Sorten ist z.B. in [Sch86, ST88, ST84, NO87, ONE89] zu finden. Ähnliche Definitionen findet man in [Hen89a, Hen89b, ST86].

Interessant ist das Verhalten von $\equiv_{beob}$ bzgl. der Übersetzung mit Signaturmorphismen $\sigma: \Sigma \to \Sigma'$.

Satz 8.15 *Ist $\sigma: \Sigma \to \Sigma'$ ein Signaturmorphismus der Gleichungslogik und beob eine Menge beobachtbaren Sorten von Σ, dann gilt*

$$\equiv_{\sigma(beob)} \subseteq \sigma(\equiv_{beob}).$$

Beweis Die Trägermengen der beobachtbaren Sorten s aus $beob$ von $A|_\sigma$ und $B|_\sigma$ sind gleich, da die Trägermengen der beobachtbaren Sorten $\sigma(s)$ von A und B gleich sind und wegen der Definition des σ–Redukts

$$(A|_\sigma)_s = A_{\sigma(s)} = B_{\sigma(s)} = (B|_\sigma)_s.$$

Zu zeigen ist, daß für einen beobachtbaren Term t aus $T_\Sigma(V_{beob})_{beob}$ und Variablenbelegungen $b_{A|_\sigma}: V \to A|_\sigma$ und $b_{B|_\sigma}: V \to B|_\sigma$, mit gleicher Belegung für Variablen beboachtbarer Sorte, die Auswertung von t in $A|_\sigma$ und in $B|_\sigma$ gleich sind

$$b^*_{A|_\sigma}(t) = b^*_{B|_\sigma}(t).$$

Die Übersetzung von t via σ ist ein beobachtbarer Term aus $T_{\Sigma'}(V_{\sigma(beob)})_{\sigma(beob)}$ und die Variablenbelegungen $b_{A|_\sigma}$ und $b_{B|_\sigma}$ definieren Variablenbelegungen der Variablen in $\sigma(t)$. Nach Voraussetzung sind die Auswertungen von $\sigma(t)$ in A und in B, in den durch $b_{A|_\sigma}$ und $b_{B|_\sigma}$ gegebenen Variablenbelegungen, gleich. Mit Lemma 2.11 erhält man dann, daß die Auswertung von t in $A|_\sigma$ und $B|_\sigma$ unter den Variablenbelegungen $b_{A|_\sigma}$ und $b_{B|_\sigma}$ ebenfalls gleich sind. ■

Für die Stabilität (s.a. Kapitel 10) von parametrisierten abstrakten Datentypen allerdings ist die Umkehrung wichtiger, d.h. folgt aus der Äquivalenz von $A|_\sigma$ und $B|_\sigma$ bzgl. $beob$ die Äquivalenz von A und B bzgl. $\sigma(beob)$? Im Gegensatz zu der Verhaltensäquivalenz (s.a. Abschnitt 8.1) gilt die Umkehrung des Satzes nicht, wie folgendes Beispiel zeigt.

Beispiel Sei Σ eine Signatur mit einer Sorte s, die beobachtbar ist ($beob = \{s\}$). Und Σ hat keine Konstanten und Operationssymbole. Sei Σ' die Signatur Σ erweitert um eine Konstante c der Sorte s. Als Signaturmorphismus $\sigma: \Sigma \to \Sigma'$ wähle die Inklusion. Die zwei Σ'–Algebren A und B haben als Trägermenge der Sorte s die zweielementige Menge $A_s = B_s = \{1, 2\}$ und A interpretiert die Konstante c als 1 ($c^A = 1$) und B die Konstante c als 2 ($c^B = 2$).

Die Algebren $A|_\sigma$ und $B|_\sigma$ sind beobachtbar äquivalent bzgl. $beob$, da die Trägermengen von $A|_\sigma$ und $B|_\sigma$ der Sorte s gleich sind und $T_\Sigma(V_{beob})_{beob}$ nur Variablen beobachtbarer Sorte enthält.

Andererseits sind A und B nicht beobachtbar äquivalent bzgl. $\sigma(beob) = \{s\}$, da nun c ein beobachtbarer Term ist, aber für jede Variablenbelegung $b_A: V \to A$ und $b_B: V \to B$

$$b^*_A(c) = 1 \neq b^*_B(c) = 2$$

ist.

Der Grund für dieses unterschiedliche Verhalten der Verhaltensäquivalenz und der Äquivalenz bzgl. beobachtbarer Sorten ist darin zu suchen, daß bei der Übersetzung von Beobachtungen via σ keine neuen Beobachtungen hinzukommenn können, auch wenn die Zielsignatur neue Operationen auf beobachtbaren Sorten enthält. Bei der Äquivalenz bzgl. beobachtbarer Sorten dagegen können diese neuen Operationen neue beobachtbare Terme zur Folge haben, deren Wert dann abhängt von der Interpretation der neuen Operationen.

Dieses Verhalten zerstört die Vorstellung in z.B. [ST84, ST85b, ST85a], daß die Äquivalenz beobachtbarer Sorten ein Spezialfall der Verhaltensäquivalenz ist, bei der die Beobachtungen alle Gleichungen (mit freien Variablen) zwischen beobachtbaren Termen sind.

Das Problem bei der Übersetzung und der Äquivalenz bzgl. beobachtbarer Sorten tritt auf, weil beobachtbare Terme über Σ' i.a. keine Beziehung zu den beobachtbaren Termen aus Σ haben. Die hinreichende Vollständigkeit von σ für eine Menge M von Σ'–Algebren und $beob$ allerdings garantiert, daß es zu jedem beobachtbaren Term t über Σ' einen beobachtbaren Term t' über Σ gibt, so daß die beiden Terme in allen Strukturen aus M gleich sind.

Definition 8.16 (Hinreichend vollständig) *Ist $\sigma: \Sigma \to \Sigma'$ ein Signaturmorphismus und beob, eine Teilmenge von sorts(Σ), eine Menge von beobachtbaren Sorten, dann ist σ hinreichend vollständig für eine Menge von Algebren $M \subseteq Alg(\Sigma')$ und beob, gdw. es zu jedem Term t aus $T_{\Sigma'}(V_{\sigma(beob)})_{\sigma(beob)}$ einen beobachtbaren Term t' aus $T_\Sigma(V_{beob})_{beob}$ gibt, so daß $t = \sigma(t')$ in allen Algebren aus M gültig ist*

$$M \models t = \sigma(t').$$

Die Definition der hinreichenden Vollständigkeit findet man ähnlich in [ST86, Gan83, Wir89, Pad85]. Man erhält z.B. die Definition aus [Gan83], wenn alle Parametersorten beobachtbar sind und $M = Mod(E')$, für eine Menge von Gleichungen $E' \subseteq GLSen(\Sigma')$, ist.

Satz 8.17 *Ist $\sigma: \Sigma \to \Sigma'$ ein Signaturmorphismus, beob $\subseteq$ sorts(Σ) ein Menge von beobachtbaren Sorten, M eine Menge von Σ'–Algebren und A und B aus M, dann ist*

$$A|_\sigma \equiv_{beob} B|_\sigma \Rightarrow A \equiv_{\sigma(beob)} B$$

wenn σ ausreichend vollständig für M und beob ist. Mit anderen Worten

$$\sigma(\equiv_{beob}) \subseteq \equiv_{\sigma(beob)},$$

wobei $\sigma(\equiv_{beob})$ auf $M \times M$ beschränkt ist.

Bei hinreichender Vollständigkeit von σ zusammen mit Satz 8.15 gilt dann

$$\sigma(\equiv_{beob}) = \equiv_{\sigma(beob)}.$$

Beweis Es ist zu zeigen, daß A_s und B_s für alle $s \in \sigma(beob)$ gleich sind, und daß für jeden beobachtbaren Term $t \in T_{\Sigma'}(V_{\sigma(beob)})_{\sigma(beob)}$ die Auswertung von t in A und in B den gleichen Wert liefern für Variablenbelegungen b_A und b_B, die den Variablen in t die gleichen Werte zuordnen.

Nach der Definition des Redukts ist $(A|_\sigma)_s = A_{\sigma(s)}$ für $s \in beob$. Da $A|_\sigma$ und $B|_\sigma$ beobachtbar äquivalent sind, sind $(B|_\sigma)_s$ und $(A|_\sigma)_s$ und damit auch $A_{\sigma(s)}$ und $B_{\sigma(s)}$ gleich.

Ist t ein beobachtbarer Term in Σ', dann gibt es wegen der hinreichenden Vollständigkeit von σ einen beobachtbaren Term t' aus $T_\Sigma(V_{beob})_{beob}$ für die $A \models t = \sigma(t')$ und $B \models t = \sigma(t')$ gilt. Nach der Definition der Erfüllbarkeit von Gleichungen muß dann unter jeder Variablenbelegung b_A und b_B die Auswertung von t gleich der Auswertung von $\sigma(t')$ sein.

Es reicht demnach zu zeigen, daß die Auswertung von $\sigma(t')$ unter b_A und b_B gleich sind. Definiere nun $b'_A: V' \to A|_\sigma$ und $b'_B: V' \to B|_\sigma$ mit

$$\begin{aligned} b'_A(v') &:= b_A(\sigma(v')) \\ \text{und } b'_B(v') &:= b_B(\sigma(v')), \end{aligned}$$

dann ist $b'_A(v') = b'_A(v')$ für v' aus $beob$, da $b_A(v) = b_B(v)$ für v aus $\sigma(beob)$ ist. Nach Voraussetzung gilt dann aber $b'^*_A(t) = b'^*_B(t)$ und mit Lemma 2.11 folgt $b^*_A(\sigma(t)) = b^*_B(\sigma(t))$. ■

Definition 8.18 (obs(beob)) *Ist beob $\subseteq$ sorts(Σ) eine Menge von beobachtbaren Sorten, dann ist folgende Menge von Beobachtungen definiert:*

$$obs(beob) := \{t = t' \mid t, t' \in T_\Sigma(V_{beob})_{beob}\}.$$

Die Menge $obs(beob)$ enthält nur Formeln aus $GLSen(\Sigma)$ und keine Formeln mit freien Variablen.

Satz 8.19 *Ist beob eine Menge von beobachtbaren Sorten aus* Σ*, dann gilt:*

$$\equiv_{beob} \subseteq \equiv_{obs(beob)}.$$

Beweis Es ist zu zeigen, daß aus $A \equiv_{beob} B$ folgt, daß für alle $t = t'$ aus $obs(beob)$ gilt

$$A \models t = t' \Leftrightarrow B \models t = t'.$$

Ist $A \models t = t'$ gegeben und $b_B: V \to B$ eine Variablenbelegung, dann gilt $b_B^*(t) = b_A^*(t)$ und $b_B^*(t') = b_A^*(t')$ für eine Variablenbelegung $b_A: V \to A$, die mit b_B auf den Variablen aus V_{beob} übereinstimmt, da t und t' beobachtbare Terme sind und $A \equiv_{beob} B$ ist. Da $A \models t = t'$ ist, erhält man dann

$$b_B^*(t) = b_A^*(t) = b_A^*(t') = b_B^*(t').$$

Entsprechend verläuft der Beweis, daß $B \models t = t'$ die Gültigkeit von $t = t'$ in A impliziert. ■

Die Rückrichtung des Satzes gilt nur dann (s.a. [ST84]), wenn die Variablen in den Gleichungen zwischen beobachtbaren Termen frei sind. In der Gleichungslogik sind sie implizit durch Allquantoren gebunden, d.h. $x = y$ ist äquivalent zu $\forall x, y : x = y$.

Kapitel 9

Zulässiger aktueller Parameter bzgl. Abstraktoren

Wie im Abschnitt 7.2 über die horizontale Komposition von Implementierungen angedeutet, müssen für die horizontale Komposition aktuelle Parameter adt_A bzgl. ihres Abstraktors α_A zulässig für einen parametrisierten abstrakten Datentypen B sein.

Definition 9.1 (zulässiger aktueller Parameter bzgl. Abstraktoren)
Ein abstrakter Datentypen adt_A ist zulässiger aktueller Parameter für einen parametrisierten abstrakten Datentyp B bzgl. des Abstraktors α_A, gdw. $Beh_{\alpha_A}(adt_A)$ ein zulässiger aktueller Parameter für B ist.

Da $adt_A \subseteq Beh_{\alpha_A}(adt_A)$ ist, ist adt_A immer dann zulässiger Parameter für B, wenn adt_A bzgl. α_A zulässiger Parameter für B ist. Untersucht wird in diesem Kapitel, unter welchen Bedingungen aus der Zulässigkeit von adt_A für B die Zulässigkeit von adt_A bzgl. eines Abstraktors α_A folgt.

Die Bedingung der Zulässigkeit bzgl. α_A

$$Beh_{\alpha_A}(adt_A)|_\rho \subseteq adt_F$$

ist erfüllt, wenn das Redukt einer zu einem Modell aus adt_A $\equiv_{\alpha_A}$–äquivalenten Struktur auf die formale Parametersignatur ein Modell von adt_F ist. Eine hinreichende Bedingung dafür ist, daß $Mod(adt_F)$ abgeschloßen gegen $\equiv_{\alpha_A}|_\rho$–Äquivalenz ist, d.h. alle zu einem Modell aus adt_F $\equiv_{\alpha_A}|_\rho$–äquivalenten Strukturen sind Modelle von adt_F.

Satz 9.2 *Ist der abstrakte Datentyp adt_A ein zulässiger aktueller Parameter für einen parametrisierten abstrakten Datentyp $B(X\colon adt_F)_\beta\colon adt_B$, dann ist adt_A zulässiger aktueller Parameter für B bzgl. des Abstraktors α_A, wenn*

$$Beh_{\alpha_A}(\rho(adt_F)) \subseteq \rho(adt_F).$$

Beweis Sei m'_A α_A–äquivalent zu einem m_A aus $Mod(adt_A)$, dann ist zu zeigen, daß $m'_A|_\rho$ in $Mod(adt_F)$ liegt.

Da adt_A zulässiger Parameter für B ist, ist $m_A|_\rho$ ein Modell von adt_F, folglich ist m_A in $Mod(\rho(adt_F))$ und da $m'_A \equiv_{\alpha_A} m_A$ gilt, ist m'_A in $Beh_{\alpha_A}(\rho(adt_F))$.

Mit der Voraussetzung $Beh_{\alpha_A}(\rho(adt_F)) \subseteq \rho(adt_F)$ ist m'_A in $Mod(\rho(adt_F))$ und damit ist $m'_A|_\rho$ in $Mod(adt_F)$. ■

Ist $adt_F = Mod(E_F)$ und sind alle Formeln in E_F beobachtbar, d.h. $E_F \subseteq cl(obs)$, dann gilt

$$\equiv_{obs}(Mod(E_F)) = Mod(E_F).$$

Als Konsequenz erhält man, daß adt_A bzgl. der Beobachtungen in obs_A zulässig ist, wenn alle Formeln in E_F im aktuellen Parameter sichtbar sind, d.h. in $cl(obs_A)$ liegen.

Satz 9.3 (Zulässigkeit bzgl. Beobachtungen) *Gegeben sei ein parametrisierter abstrakter Datentyp $B(X\!:adt_F)_\beta\!:adt_B$ mit $Mod(adt_F) = Mod(E_F)$. Ein abstrakter Datentyp adt_A ist zulässiger aktueller Parameter für B bzgl. der Beobachtungen in obs_A wenn*

1. *adt_A zulässiger Parameter für B*
2. *und $\rho(E_F) \subseteq cl(obs_A)$ ist.*

Beweis Es ist zu zeigen, daß für ein m'_A aus $Mod(Beh_{obs_A}(adt_A))$ das Redukt von m'_A auf Σ_F in $Mod(adt_F)$ liegt.

Ist m'_A in $Mod(Beh_{obs_A}(adt_A))$, dann gibt es ein m_A aus $Mod(adt_A)$ sodaß m'_A und m_A obs_A–äquivalent sind.

Da $adt_A|_\rho \subseteq adt_F$ und $Mod(adt_F) = Mod(E_F)$ ist, gilt daß in $m_A|_\rho$ alle Formeln aus E_F gültig sind. Da die Formeln in E_F beobachtbar sind, d.h. in $cl(obs)$ liegt, sind die Formeln aus E_F in $m'_A|_\rho$ ebenfalls gültig und damit liegt $m'_A|_\rho$ in adt_F. ■

Enthält z.B. wie bei der parametrisierten Spezifikation $Stack(X\!:Elem)\!:Stack$ die formale Parameterspezifikation keine Formeln, dann ist jeder zulässige aktuelle Parameter auch zulässiger aktueller Parameter bzgl. beliebigem obs_A. Insbesondere ist $Stack(Nat)$ zulässiger aktueller Parameter bzgl. obs_A, auch wenn obs_A keine Gleichheiten auf der Sorte *stack* enthält.

Für die Äquivalenz bzgl. beobachtbarer Sorten erhält man die Zulässigkeit eines Parameters bzgl. einer Menge von beobachtbaren Sorten $beob_A$, wenn $beob_A$ alle formalen Parametersorten enthält.

Satz 9.4 (Zulässigkeit bzgl. beobachtbarer Sorten) *Gegeben sei ein parametrisierter abstrakter Datentyp $B(X\!:adt_F)_\beta\!:adt_B$ mit $Mod(adt_F) = Mod(E_F)$. Ein abstrakter Datentyp adt_A ist zulässiger aktueller Parameter für B bzgl. der beobachtbaren Sorten in $beob_A$ wenn*

1. *adt_A zulässiger Parameter für B,*
2. *und $\rho(sorts(Sign(adt_F))) \subseteq beob_A$ ist.*

Beweis Ist $m'_A \in Mod(Beh_{beob_A}(adt_A))$, dann ist zu zeigen, daß $m'_A|_\rho$ in $Mod(adt_F)$ ist. Es gibt ein $m_A \in Mod(adt_A)$ mit $m'_A \equiv_{beob_A} m_A$. Da $\rho(sorts(Sign(adt_F)))$ eine Teilmenge von $beob_A$ ist, sind m'_A und m_A auch äquivalent bzgl. $\rho(sorts(Sign(adt_F)))$ und damit sind $m'_A|_\rho$ und $m_A|_\rho$ äquivalent bzgl. $sorts(Sign(adt_F))$.

Aus der Definition 8.14 der Äquivalenz bzgl. beobachtbarer Sorten folgt dann, daß $m'_A|_\rho$ und $m_A|_\rho$ gleich sind und da $m_A|_\rho$ in $Mod(adt_F)$ ist, da $m_A \in Mod(adt_A)$ ist, ist $m'_A|_\rho$ ebenfalls in $Mod(adt_F)$. ■

Allerdings ist dabei $Stack(Nat)$, bei nicht beobachtbarer Sorte *stack*, kein zulässiger aktueller Parameter für $Stack(X\!:Elem)\!:Stack$ bzgl. $beob_A = \{nat\}$. Das folgende Lemma führt die Zulässigkeit bzgl. eines Abstraktors auf die Zulässigkeit bzgl. eines Abstraktor mit schwächerer Äquivalenzrelation zurück.

Lemma 9.5 *Ist der abstrakte Datentyp adt_A ein bzgl. α_A zulässiger aktueller Parameter für $B(X\colon adt_F)_\beta\colon adt_B$, dann ist adt_A ein bzgl. α'_A zulässiger aktueller Parameter für B, wenn*

$$\equiv_{\alpha'_A} \subseteq \equiv_{\alpha_A}$$

gilt.

Beweis Es ist zu zeigen, daß $Beh_{\alpha'_A}(adt_A)|_\rho \subseteq adt_F$ gilt. Da $\equiv_{\alpha'_A}$ eine Teilmenge von $\equiv_{\alpha_A}$ ist, gilt auch

$$Beh_{\alpha'_A}(adt_A) \subseteq Beh_{\alpha_A}(adt_A)$$

und da adt_A ein bzgl. α_A zulässiger Parameter von B ist, erhält man

$$Beh_{\alpha'_A}(adt_A)|_\rho \subseteq adt_F.$$

■

Damit wird es möglich die Zulässigkeit bzgl. der Äquivalenz bzgl. beobachbarer Sorten auf die Zulässigkeit der Verhaltensäquivalenz zurückzuführen.

Satz 9.6 (Zulässigkeit bzgl. beobachtbarer Sorten) *Gegeben sei ein parametrisierter abstrakter Datentyp $B(X\colon adt_F)_\beta\colon adt_B$ mit $Mod(adt_F) = Mod(E_F)$. Ein abstrakter Datentyp adt_A ist zulässiger aktueller Parameter für B bzgl. der beobachtbaren Sorten in $beob_A$ wenn*

1. *adt_A zulässiger Parameter für B,*
2. *und $\rho(E_F) \subseteq cl(obs(beob_A))$ ist.*

Beweis Nach Satz 8.19 ist $\equiv_{beob_A}$ Teilmenge von $\equiv_{obs(beob_A)}$ und zusammen mit den Voraussetzungen ist nach Satz 9.3 adt_A zulässiger Parameter bzgl. der Beobachtungen in $obs(beob_A)$. Mit Lemma 9.5 erhält man dann, daß adt_A ein bzgl. $beob_A$ zulässiger Parameter für B ist. ■

In dem Stackbeispiel bedeutet das, daß $Stack(Nat)$, auch bei unbeobachtbarer Sorte *stack*, zulässiger Parameter für $Stack(X\colon Elem)\colon Stack$ bzgl. $beob_A = \{nat\}$ ist, da $Stack(Nat)$ zulässig bzgl. beliebigen Beobachtungen, also auch denen aus $obs(beob_A)$, ist.

Kapitel 10

Stabilität

Der Begriff Stabilität stammt aus [Sch86] und bezeichnet dort eine hinreichende und notwendige Bedingung für die Komposition von Implementierugen. Der Begriff Stabilität stammt aus einem etwas anderen Implementierungsbegriff, wie den hier verwendeten. Ein parametrisierter abstrakter Datentyp[1] B wird durch B' implementiert, wenn für jeden zulässigen Parameter adt_A und jedem $m_A \in Mod(adt_A)$ gilt

$$B'_R(\equiv_{\alpha_A}(m_A)) \subseteq \equiv_{\alpha'}(B_R(m_A)).$$

Dieser Implementierungsbegriff wird universelle Implementierung genannt [Sch86, ST88] und läßt sich in die einfache Implementierung

$$B'_R(m_A) \subseteq \equiv_{\alpha'} B_R(m_A)$$

und die Stabilität von B' zerlegen

$$B'_R(\equiv_{\alpha_A}(m_a)) \subseteq \equiv_{\alpha'}(B'_R(m_A)).$$

Die einfache Implementierung entspricht fast der Abstraktorimplementierung. Der einzige Unterschied besteht darin, daß bei der einfachen Implementierung eine Struktur aus $B'_R(m_A)$ äquivalent zu einer Struktur aus $B_R(m_A)$ sein muß, während bei der Abstraktorimplementierung die Äquivalenz zu irgend einem Modell aus $B(adt_A)$ ausreicht.

Das Analogon zur Stabilität aus [Sch86, ST88], ist das Erhalten der Abstraktoräquivalenz [ST86]. Die Definition der Erhaltung der Abstraktoräquivalenz stammt aus [ST86] und findet sich als Monotonie in [Hen89b] wieder.

Definition 10.1 (Erhalten der Abstraktor–Äquivalenz) *Ein parametrisierter abstrakter Datentyp B* erhält die α_A–Äquivalenz bzgl. des Abstraktors α_B *für adt_A, gdw.*

1. *adt_A ein bzgl. α_A zulässiger Parameter für B*
2. *und $B(Beh_{\alpha_A}(adt_A)) \subseteq Beh_{\alpha}(B(adt_A))$ ist,*

wobei α_A ein Abstraktor für adt_A, α_B ein Abstraktor für adt_B und $\alpha := \rho'(\alpha_B)$ ein Abstraktor für $B(adt_A)$ ist.

Definition 10.2 (Stabilität) *Ein parametrisierter abstrakter Datentyp B ist* stabil *für adt_A bzgl. der Abstraktoren α_A und α_B, gdw.*

[1] in [Sch86] Zellen genannt. Die Zellen sind wie parametrisierte abstrakte Datentypen definiert.

1. *adt_A ein bzgl. α_A zulässiger aktueller Parameter ist*

2. *und für alle m_A aus $Mod(adt_A)$*

$$B_R(\equiv_{\alpha_A}(m_A)) \subseteq \equiv_{\alpha}(B_R(m_A))$$

gilt,

wobei α_A ein Abstraktor für adt_A, α_B ein Abstraktor für adt_B und $\alpha := \rho'(\alpha_B)$ ein Abstraktor für $B(adt_A)$ ist.

Satz 10.3 *Ist B ein parametrisierter abstrakter Datentyp und B stabil für adt_A bzgl. der Abstraktoren α_A und α_B, dann erhält B die Verhaltensäquivalenz für adt_A bzgl. α_A und α_B.*

Beweis Es ist zu zeigen, daß die Modelle von $B(Beh_{\alpha_A}(adt_A))$ auch in $Beh_{\alpha}(B(adt_A))$ liegen, wobei $\alpha = \rho'(\alpha_B)$ ist.

Ist also m' aus $Mod(B(Beh_{\alpha_A}(adt_A)))$, dann gibt es ein $m_A \in Mod(adt_A)$ mit $m'|_{\beta'} \equiv_{\alpha_A} m_A$ und wegen der Definition von B_R und Beh_{α_A} gilt

$$m' \in B_R(Beh_{\alpha_A}(m_A)).$$

Wegen der Stabilität von B für adt_A bzgl α_A und α_B ist

$$m' \in Beh_{\alpha}(B_R(m_A))$$

und da m_A in $Mod(adt_A)$ ist, ist

$$m' \in Beh_{\alpha}(B(adt_A)).$$

■

Satz 10.4 *Erhält ein parametrisierter abstrakter Datentyp B die α_A–Äquivalenz bzgl. des Abstraktors α_B für alle bzgl. α_A zulässigen aktuellen Parameter adt_A, dann ist B stabil für adt_A bzgl. α_A und α_B.*

Beweis Ist adt_A ein bzgl. α_A zulässiger Parameter von B, dann ist jeder abstrakte Datentyp adt'_A mit $Sign(adt'_A)$ gleich $Sign(adt_A)$ und $Mod(adt'_A)$ gleich $\{m_A\}$ für ein beliebiges m_A aus $Mod(adt_A)$ ein bzgl. α_A zulässiger Parameter.

Erhalte B die α_A–Äquivalenz für adt'_A. Wegen der Definition von B_R ist

$$|Mod(B(adt'_A))| = B_R(m_A)$$

und

$$|Mod(B(Beh_{\alpha_A}(adt'_A)))| = B_R(Beh_{\alpha_A}(m_A))$$

Damit kann man die Bedingung für die Erhaltung der α_A –Äquivalenz B umschreiben zu

$$B_R(Beh_{\alpha_A}(m_A)) \subseteq Beh_{\alpha}(B_R(m_A)).$$

Da diese Bedingung für alle m_A aus $Mod(adt_A)$ gilt, ist das die Stabilität von B für adt_A bzgl. α_A und α_B. ■

Wie bei der Konsistenz und Eindeutigkeit von parametrisierten abstrakten Datentypen läßt sich unter der Voraussetzung der Existenz eindeutiger amalgamierter Summen, die Stabilität für den formalen Parameter auf die Stabilität für alle aktuellen Parameter erweitern.

Satz 10.5 (Extension Lemma: Stabilität) *Wenn in der Instiution $\mathcal{I}$ eindeutige amalgamierte Summen existieren, dann gilt:*

Ein parametrisierter abstrakter Datentyp B ist für adt_A bzgl. der Abstraktoren α_A und α_B stabil, wenn

1. *adt_A ein bzgl. α_A zulässiger Parameter für B,*
2. *B stabil für adt_F bzgl. α_F und α_B*
3. *und $(\equiv_{\alpha_A})|_\rho \subseteq \equiv_{\alpha_F}$ ist.*

Beweis Es ist zu zeigen, daß für alle $m_A \in Mod(adt_A)$

$$B_R(\equiv_{\alpha_A}(m_A)) \subseteq \equiv_\alpha(B_R(m_A))$$

gilt, wobei $\equiv_\alpha := \rho'(\equiv_{\alpha_B})$ ist.

Sei also m' in $B_R(\equiv_{\alpha_A}(m_A))$, dann ist $m'_A := m'|_{\beta'}$ α_A–äquivalent zu m_A, dann aber auch laut Voraussetzung ist $m'_F := m'_A|_\rho$ α_F–äquivalent zu $m_F := m_A|_\rho$.

Da nun B stabil für adt_F ist und $m'_B := m'|_{\rho'}$ in $B_R(\equiv_{\alpha_F}(m_F))$ liegt, da $m'_B|_\beta = m'_F$ ist, existiert ein m_B aus $B_R(m_F)$ und m_B ist α_B–äquivalent zu m'_B. Die Struktur $m := m_A +_{m_F} m_B$ liegt dann in $B_R(m_A)$ und es gilt, da wegen $m_B \equiv_{\alpha_B} m'_B$ auch m Æ$\rho'(\equiv_{\alpha_B})$ m' ist, daß m' in $\equiv_\alpha(B_R(m_A))$ liegt. ■

Stabilität erhält man, wenn für zwei $\equiv_{\alpha_A}$–äquivalente Modelle aus $Mod(Sign(adt_A))$ ihre Erweiterungen aus $Mod(B(adt_A))$ $\rho'(\equiv_{\alpha_B})$–äquivalent sind. Die Konsistenz von B garantiert, daß die Erweiterungen der Modelle existieren.

Satz 10.6 *Ein parametrisierter abstrakter Datentyp B ist bzgl. α_A und α_B stabil für adt_A, wenn*

1. *adt_A ein bzgl. α_A zulässiger Parameter für B,*
2. *B konsistent für adt_A*
3. *und $\beta'(\equiv_{\alpha_A}) \subseteq \rho'(\equiv_{\alpha_B})$ ist, wobei $\beta'(\equiv_{\alpha_A})$ auf $|Mod(B(adt_A))| \times |Mod(B(adt_A))|$ eingeschränkt ist.*

Beweis Es ist zu zeigen, daß für alle m_A aus $Mod(adt_A)$

$$B_R(\equiv_{\alpha_A}(m_A)) \subseteq \equiv_\alpha(B_R(m_A))$$

gilt, wobei $\equiv_\alpha := \rho'(\equiv_{\alpha_B})$ ist.

Wegen der Konsistenz von B für adt_A gibt es mindestens ein $m \in B_R(m_A)$. Sei nun $m' \in B_R(\equiv_{\alpha_A}(m_A))$, dann reicht es zu zeigen, daß $m' \equiv_\alpha m$ gilt, damit $m' \in \equiv_\alpha(B_R(m_A))$ ist. Zusammen mit der Voraussetzung und der Tatsache, daß $m'|_{\beta'} \equiv_{\alpha_A} m_A = m|_\beta$ ist, erhält man dann $m' \equiv_\alpha m$. ■

Eine Variante dieses Satzes ist, daß für zwei Strukturen aus $Mod(Sign(adt_F))$, die $(\equiv_{\alpha_A})|_\rho$–äquivalent sind, alle Erweiterungen aus $Mod(adt_B)$ $\equiv_{\alpha_B}$–äquivalent sind.

Satz 10.7 *Ein parametrisierter abstrakter Datentyp B ist bzgl. α_A und α_B stabil für adt_A, wenn*

1. *adt_A ein bzgl. α_A zulässiger Parameter für B,*
2. *B konsistent für adt_A*
3. *und $\beta((\equiv_{\alpha_A})|_\rho) \subseteq \equiv_{\alpha_B}$ ist, wobei $\beta((\equiv_{\alpha_A})|_\rho)$ auf $|Mod(adt_B)| \times |Mod(adt_B)|$ eingeschränkt ist.*

Beweis Um Satz 10.6 anzuwenden muß aus der Bedingung

$$\beta((\equiv_{\alpha_A})|_\rho) \subseteq \equiv_{\alpha_B}$$

die Bedingung

$$\beta'(\equiv_{\alpha_A}) \subseteq \rho'(\equiv_{\alpha_B})$$

folgen. Sei also $m\ \beta'(\equiv \alpha_A)\ m'$, dann ist zu zeigen daß $m\ \rho'(\equiv \alpha_B)\ m'$ gilt.

Aus $m\ \beta'(\equiv \alpha_A)\ m'$ folgt, daß $(m|_{\beta'})|_\rho$ und $(m'|_{\beta'})|_\rho$ äquivalent bzgl. $(\equiv_{\alpha_A})|_\rho$ sind, also sind auch $m|_{\rho'}$ und $m'|_{\rho'}$ äquivalent bzgl. $\beta((\equiv_{\alpha_A})|_\rho)$ und damit laut Vorausssetzung bzgl. $\equiv_{\alpha_B}$. Daraus folgt dann

$$m\ \rho'(\equiv \alpha_B)\ m'.$$

■

Aus dem Korollar 8.7, das für eine Menge von Beobachtungen *obs* und einen Signaturmorphismus σ aussagt

$$\sigma(\equiv_{obs}) = \equiv_{\sigma(obs)},$$

ist die Bedingung der Stabilität bei Verhaltensäquivalenz erfüllt wenn $\rho'(obs_B) \subseteq \beta'(obs_A)$ ist, denn dann gilt

$$\beta'(\equiv_{obs_A}) = \equiv_{\beta'(obs_A)} \subseteq \equiv_{\rho'(obs_B)} = \rho'(\equiv_{obs_B}).$$

Satz 10.8 (Stabilität bzgl. Beobachtungen) *Ein parametrisierter abstrakter Datentyp B ist stabil für adt_A bzgl. der Beobachtungen obs_A und obs_B, wenn*

1. *adt_A ein bzgl. obs_A zulässiger Parameter,*
2. *B konsistent für adt_A*
3. *und $\rho'(obs_B) \subseteq \beta'(cl(obs_A))$ ist.*

Beweis Nach Satz 10.6 reicht es zu zeigen, daß $\beta'(\equiv_{obs_A}) \subseteq \rho'(\equiv_{obs_B})$ ist. Das gilt, da

$$\begin{aligned} \beta'(\equiv_{obs_A}) &= \beta'(\equiv_{cl(obs_A)}) \\ &= \equiv_{\beta'(cl(obs(A)))} \end{aligned}$$

und

$$\rho'(\equiv_{obs_B}) = \equiv_{\rho'(obs_B)}$$

ist und wegen $\rho'(obs_B) \subseteq \beta'(cl(obs_A))$. ■

Das Problem der Bedingung $\rho'(obs_B) \subseteq \beta'(cl(obs_A))$ ist, daß alle Operationen, die im Rumpf neu hinzukommen und Resultate im Parameter haben, nicht mehr sichtbar sind. Zum Beispiel kann man nicht mehr das oberste Element eines Stacks beobachten, denn $top(s) = e$ kann nicht in obs_B enthalten sein, da *top* keine Operation des formalen Parameters ist.

Bei der Verhaltensäquivalenz ist man also gezwungen neue Beobachtungen im Rumpf zuzulassen. Lassen sich zu diesen Beobachtungen äquivalente Beobachtungen im formalen Parameter finden, erhält man Stabilität. Diese Bedingung findet sich in [ST86] als Erweiterung der hinreichenden Vollständigkeit auf beliebigen Institutionen.

Die Stabilität bei beobachtbaren Sorten ist dann gegeben, wenn im Rumpf keine neuen beobachtbaren Sorten zu denen im aktuellen Parameter hinzukommen, und wenn es zu jedem beobachtbaren Term des Rumpfes einen äquivalenten beobachtbaren Term des Parameters gibt, d.h. $\beta: adt_F \to adt_B$ hinreichend vollständig für $Mod(adt_B)$ ist.

Satz 10.9 (Stabilität bzgl. beobachtbarer Sorten)
Ein parametrisierter abstrakter Datentyp $B(X: adt_F)_\beta: adt_B$ ist stabil für adt_A bzgl. der beobachtbaren Sorten $beob_A$ und $beob_B$, wenn

1. *adt_A ein bzgl. $beob_A$ zulässiger Parameter,*
2. *B konsistent für adt_A,*
3. *$beob_B \subseteq \beta(beob_F)$*
4. *und $Sign(\beta)$ hinreichend vollständig für $Mod(adt_B)$ und $beob_F$ ist,*

wobei $beob_F := \{s \in sorts(Sign(adt_F)) \mid \rho(s) \in beob_A\}$ ist.

Beweis Nach Satz 10.7 reicht es zu zeigen, daß $\beta((\equiv_{\alpha_A})|_\rho) \subseteq \equiv_{obs_B}$ ist. Da $\rho(beob_F)$ Teilmenge von $beob_A$ ist, folgt aus $m_A \equiv_{beob_A} m'_A$ daß $m_A|_\rho \equiv_{beob_F} m'_A|_\rho$ und damit daß

$$(\equiv_{beob_A})|_\rho \subseteq \equiv_{beob_F}$$

gilt. Wegen der hinreichenden Vollständigkeit von $Mod(adt_B)$ erhält man

$$\beta(\equiv_{beob_F}) = \equiv_{\beta(beob_F)}$$

und somit

$$(\equiv_{beob_A})|_\rho \subseteq \equiv_{\beta(beob_F)}.$$

■

Daß der Rumpf keine neuen beobachtbaren Sorten enthält entspricht der Vorstellung in [Sch86, ST85a, ST88, Gan83], daß einen nicht zu interessieren hat, wie die Elemente der neuen Sorten im Rumpf auszusehen haben. Gerade von Implementierungsdetails möchte man ja abstrahieren (Datenabstraktion).

Die beobachtbaren Sorten in $beob_B$ sind abhängig von den beobachtbaren Sorten in $beob_A$. Wird eine sichtbare aktuelle Parametersorte an eine formale Parametersorte gebunden, kann und sollte sie in $beob_B$ enthalten sein, ist die aktuelle Parametersorte dagegen nicht sichtbar, dann darf sie nicht in $beob_B$ auftauchen. Im *Stack*–Beispiel ist bei $Stack(Nat)$ $beob_B = \{bool, nat\}$ bei $Stack(Stack(Nat))$ dagegen nur $beob_B = \{bool\}$.

Da die hinreichende Vollständigkeit von $beob_B$ abhängt, ist jedesmal wenn $beob_A$ Einfluß auf $beob_B$ hat, die hinreichende Vollständigkeit für die Stabilität zu überprüfen.

Literaturverzeichnis

[EGL89] Hans-Dieter Ehrich, Martin Gogolla, and Udo Walter Lipeck. *Algebraische Spezifikation abstrakter Datentypen (Algebraic Specification of Abstract Datatypes).* B. G. Teubner Stuttgart, 1989.

[Ehr89] Hartmut Ehrig. A categorical concept of constraints for algebraic specifications. In H. Ehrig, H. Herrlich, H.-J. Kreowski, and G. Preuß, editors, *Categorical Methods in Computer Science*, number 393 in LNCS. Springer Verlag, 1989.

[EM85] Hartmut Ehrig and Bernd Mahr. *Fundamentals of Algebraic Specification 1: Equations and initial Semantics.* Number 6 in EATCS Monographs on Theoretical Computer Science. Springer, 1985.

[Gan83] Harald Ganzinger. Parameterized specifications: Parameter passing and implementation with respect to observability. *ACM Transaction on Programming Languages*, 5(3):318–354, July 1983.

[GB90] J. A. Goguen and R. Burstall. INSTITUTIONS: Abstract model theory for specification and programming, 1990.

[Hen89a] Rolf Hennicker. Beobachtungsorientierte Spezifikationen. Technical Report MIP–8924, Universität Passau, August 1989.

[Hen89b] Rolf Hennicker. Implementation of parameterized observational specifications. In *Proc. CAAP 89*, 1989.

[Mac88] Saunders Mac Lane. *Categories for the working mathematician.* Graduated Texts in Mathematics. Springer, fourth edition, 1988.

[NO87] Ma. P. Nivela and F. Orejas. Behavioural semantics for algebraic specification languages. In D. Sannella and A. Tarlecki, editors, *Recent Trends in Data Type Specification*, number 332 in LNCS, pages 184–207. Springer, September 1987.

[ONE89] F. Orejas, Ma. P. Nivela, and H. Ehrig. Semantical constructions for categories of behavioural specifications. In H. Ehrig, H. Herrlich, H.-J. Kreowski, and G. Preuß, editors, *Categorical methods in Computer Science with aspects from Topology*, pages 220–243. Springer, 1989.

[OSC89] F. Orejas, V. Sacristán, and S. Clerici. Development of algebraic specifications with constraints. In H. Ehrig, H. Herrlich, H.-J. Kreowski, and G. Preuß, editors, *Categorical methods in Computer Science with aspects from Topology*, pages 36–49. Springer, 1989.

[Pad85] Peter Padawitz. Towards a proof theory of parameterized specifications. Bericht 8503, Universität Passau, 1985.

[Sch86] Oliver Schoett. *Data Abstraction and the Correctness of Modular Programming.* PhD thesis, University of Edinburgh, 1986.

[SST90] Donald Sannella, Stefan Sokolowski, and Andrzej Tarlecki. Toward formal development of programs from algebraic specifications: parameterization revisited, 1990.

[ST84] Donald Sannella and Andrzej Tarlecki. On observational equivalence and algebraic specification. Technical Report CSR-172-84, University of Edinburgh, 1984.

[ST85a] Donald Sannella and Andrzej Tarlecki. Extended ML: an institution–independent framework for formal program development. In *Workshop on Category Theory and Computer Programming*, number 240 in LNCS, pages 364–389. Springer, 1985.

[ST85b] Donald Sannella and Andrzej Tarlecki. Specifications in an arbitrary institution. Draft 1, March 1985.

[ST86] Donald Sannella and Andrzej Tarlecki. Toward formal development of programs from algebraic specifications: Implementation revisited. LFCS Report Series ECS–LFCS–86–17, Laboratory for Foundations of Computer Science, University of Edinburgh, December 1986.

[ST88] Donald Sannella and Andrzej Tarlecki. Toward formal development of ML programs: Foundations and methodology. draft (short version), September 1988.

[Tar84a] A. Tarlecki. Abstract algebraic institutions which strongly admit inital semantics. Edinburgh, 1984.

[Tar84b] A. Tarlecki. On the existence of free models in abstract algebraic institutions. Edinburgh, 1984.

[Tar84c] Andrzej Tarlecki. Quasi—varieties in abstract algebraic institutions. Report CSR–173–84, Department of Computer Science Univeristy of Edinburgh, 1984.

[WE85] E. G. Wagner and H. Ehrig. Canonical constraints for parameterized data types. Research Report RC 11248 (50666), IBM Thomas J. Watson Research Center, Yorktown Heights, New York 10598 / USA, Jul 1985.

[Wir89] Martin Wirsing. Algebraic specification. Technical Report MIP–8914, Fakultät für Mathematik und Informatik Universität Passau, June 1989.